

NASA Contractor Report 189632, Volume II

**Advanced Information Processing System:
The Army Fault Tolerant Architecture
Conceptual Study**

**Volume II- Army Fault Tolerant Architecture
Design and Analysis**

**R. E. Harper, L. S. Alger, C. A. Babikyan, B. P. Butler, S.
A. Friend, R. J. Ganska, J. H. Lala, T. K. Masotto, A. J.
Meyer, D. P. Morton, G. A. Nagle, C. E. Sakamaki**

**The Charles Stark Draper Laboratory, Inc.
Cambridge, MA**

**Contract NAS1-18565
July 1992**



**National Aeronautics and
Space Administration**

**Langley Research Center
Hampton, Virginia 23665-5225**

(NASA-CR-189632-Vol-2) ADVANCED
INFORMATION PROCESSING SYSTEM: THE
ARMY FAULT TOLERANT ARCHITECTURE
CONCEPTUAL STUDY. VOLUME 2: ARMY
FAULT TOLERANT ARCHITECTURE DESIGN
AND ANALYSIS (Draper (Charles
Stark) Lab.) 430 p

N92-33101

Unclass

G3/62 0116472

1N-62
Vol. 2
116472
p- 430

This page intentionally left blank.

Executive Summary

Digital computing systems needed for Army programs such as the Computer-Aided Low Altitude Helicopter Flight Program and the Armored Systems Modernization (ASM) vehicles may be characterized by high computational throughput and input/output bandwidth, hard real-time response, high reliability and availability, and maintainability, testability, and producibility requirements. In addition, such a system should be affordable to produce, procure, maintain, and upgrade.

To address these needs the Army Fault Tolerant Architecture (AFTA) is being designed and constructed under a three-year program comprising the Conceptual Study, Detailed Design and Fabrication, and Demonstration and Validation phases. This report describes the results of the Conceptual Study phase of the AFTA development. The scope of the Conceptual Study was quite broad and covered topics ranging from mission requirements to architectural synthesis and analysis to life cycle cost modeling.

AFTA is a militarized version of the Fault Tolerant Parallel Processor (FTPP) developed by the Charles Stark Draper Laboratory, Inc. AFTA is a hard-real-time Byzantine resilient parallel processor which is programmed in the Ada language. It supports testability and redundancy management strategies which permit the dynamic reconfiguration of processing sites to enhance sortie availability and mission reliability. It is composed largely of Non-Developmental Items to reduce the development risk and cost and to facilitate upgrades. Extensive analytical models and predictive verification and validation techniques are provided with AFTA to allow application designers to engineer a configuration for specific missions with a high degree of confidence that the fielded configuration will meet the mission requirements. As a part of AFTA, a fault tolerant data bus (FTDB) is being developed to provide a highly reliable, fault tolerant networking system between AFTA and other digital systems. The conceptual design of the FTDB covers many aspects of network design, including media technology, media access control, topology, routing, OSI protocol stacks, and fault detection and recovery. In addition to these traditional network topics, the FTDB also encompasses techniques from the area of fault-tolerance, including Byzantine resilience and authentication protocols.

AFTA's architectural theory of operation, the AFTA hardware architecture and components, and the architecture of the AFTA Operating System have been defined during the Conceptual Study, as well as a test and maintenance strategy for use in fielded AFTA installations. A format has been developed for representing mission requirements in a manner suitable for first-order AFTA sizing and analysis. Preliminary requirements have been obtained for two Army missions: a rotary winged aircraft mission and a ground vehicle mission. An approach to be used in reducing the probability of AFTA failure due to common-mode faults has been developed, as have analytical models for AFTA performance, reliability, availability, life cycle cost, weight, power, and volume. A plan has been developed for verifying and validating key AFTA concepts during the Dem/Val phase, especially those which cannot be cost-effectively validated by accelerated life cycle testing. The analytical models and partial Army mission requirements developed under the Conceptual Study have been used to evaluate AFTA configurations for the two selected Army missions. To assist in documentation and reprourement of AFTA components, VHDL is used to describe and design AFTA's developmental hardware. Finally, the requirements, architecture, and operational theory of the AFTA Fault Tolerant Data Bus have been defined and described.

The next phase of the development has begun and will result in a Brassboard AFTA for demonstration and validation.

This page intentionally left blank.

Table of Contents

Executive Summary	iii
Table of Contents	v
List of Figures	xv
List of Tables	xix
Introduction to Volumes I and II	xxi
4. AFTA Hardware Architecture	4-1
4.1. AFTA Physical Configuration	4-1
4.2. AFTA Virtual Configuration	4-2
4.3. AFTA Functional Overview	4-3
4.4. AFTA Network Element	4-6
4.4.1. Network Element Addressing Convention	4-6
4.4.2. Network Element Functional Description	4-8
4.4.2.1. Data Exchange Primitives	4-9
4.4.2.1.1. Class 0	4-10
4.4.2.1.2. Class 1	4-10
4.4.2.1.3. Class 2	4-11
4.4.2.1.4. Broadcasts	4-11
4.4.2.2. Configuration Table Updates	4-11
4.4.2.3. Initial Synchronization	4-12
4.4.2.4. Transient NE Recovery	4-14
4.4.2.5. Voted Resets/Monitor Interlocks	4-15
4.4.2.6. Syndrome Reports	4-16
4.4.2.7. Timestamps	4-18
4.4.2.8. NE Debug Commands	4-19
4.4.3. Network Element Programming Reference	4-20
4.4.3.1. Processor/Network Element Interface	4-21
4.4.3.2. Memory Map	4-21
4.4.3.2.1. Data Segment	4-21
4.4.3.2.1.1. Outgoing (Transmit) Buffering	4-22
4.4.3.2.1.2. Incoming (Receive) Buffering	4-23
4.4.3.2.1.3. Information Block Fields	4-24
4.4.3.2.1.3.1. Class	4-24
4.4.3.2.1.3.2. ToVID	4-26
4.4.3.2.1.3.3. FromVID	4-26
4.4.3.2.1.3.4. User Field	4-26
4.4.3.2.1.3.5. Vote errors	4-26
4.4.3.2.1.3.6. Clock Errors	4-27
4.4.3.2.1.3.7. Link Errors	4-27
4.4.3.2.1.3.8. OBNE timeout	4-27
4.4.3.2.1.3.9. IBNF timeout	4-28
4.4.3.2.1.3.10. Scoreboard Vote Error	4-28
4.4.3.2.1.3.11. Timestamp	4-29
4.4.3.2.2. Buffer Manager	4-31
4.4.3.3. Packet Formats	4-31
4.4.3.3.1. Data Packet	4-31
4.4.3.3.2. CT Update Packet	4-34
4.4.3.3.3. Transient NE Recovery Packet	4-35
4.4.3.3.4. Voted Reset Packet	4-36
4.5. AFTA Component Physical Descriptions	

4.5.1.	Processing Element (PE) Characteristics.....	4-41
4.5.2.	Network Element (NE) Characteristics.....	4-47
4.5.2.1.	Network Element Overview.....	4-47
4.5.2.1.1.	VMEbus Interface.....	4-48
4.5.2.1.2.	Network Element Data Paths	4-49
4.5.2.1.3.	Inter-FCR Communication System	4-51
4.5.2.1.4.	Fault-Tolerant Clock.....	4-52
4.5.2.1.5.	Global Controller.....	4-53
4.5.2.1.6.	Scoreboard.....	4-53
4.5.2.2.	Network Element Physical Characteristics	4-55
4.5.2.2.1.	Circuit Board Layout	4-55
4.5.2.2.2.	Military Qualification of Baseline Network Element.....	4-57
4.5.2.3.	Effect of Implementation Technology on Network Element Physical Characteristics.....	4-60
4.5.2.3.1.	RISC Processor Scoreboard.....	4-60
4.5.2.3.2.	FPGA Implementation.....	4-61
4.5.2.3.3.	High-End Network Element.....	4-62
4.5.3.	Input/Output Controller (IOC) Characteristics.....	4-67
4.5.4.	Power Conditioner (PC) Characteristics	4-68
4.5.5.	Cooling System	4-70
5.	AFTA Software Architecture	5-1
5.1.	Overview	5-1
5.2.	System Specification and Initialization	5-2
5.2.1.	Virtual Group Configuration	5-3
5.2.2.	Rate Group Task Configuration.....	5-7
5.3.	Rate Group Tasking Services	5-10
5.3.1.	Rate Group Tasking Initialization.....	5-11
5.3.2.	Rate Group Dispatcher	5-13
5.3.3.	Rate Group Tasks.....	5-15
5.4.	Time Management	5-18
5.4.1.	Time Management Initialization	5-18
5.4.2.	Time Management Operation	5-19
5.5.	Communication Services	5-20
5.5.1.	Message and Packet Structure.....	5-21
5.5.2.	Communication Services Initialization	5-24
5.5.3.	Message Transmission	5-30
5.5.4.	Message Reception	5-32
5.6.	Fault Detection, Identification and Recovery.....	5-37
5.6.1.	System and Test Modes.....	5-37
5.6.2.	Off-Line Fault, Detection, Isolation and Recovery	5-38
5.6.3.	Local Fault Detection, Isolation and Recovery	5-39
5.6.4.	System Fault Detection, Isolation and Recovery.....	5-39
5.6.5.	Operational Modes.....	5-39
5.6.6.	Fault Detection Mechanisms.....	5-40
5.6.6.1.	Enumeration of Mechanisms.....	5-41
5.6.6.1.1.	Processor Self Tests	5-41
5.6.6.1.1.1.	CPU Tests.....	5-41
5.6.6.1.1.2.	Cache Tests	5-41
5.6.6.1.1.3.	Memory Tests.....	5-42
5.6.6.1.1.4.	MMU Tests	5-42
5.6.6.1.1.5.	I/O Tests.....	5-44
5.6.6.1.1.6.	Miscellaneous Tests.....	5-44
5.6.6.1.2.	Network Element Self Tests	5-44
5.6.6.1.2.1.	Processor-Network Element Interface	5-44

5.6.6.1.2.2.	Network Element Data Paths.....	5-45
5.6.6.1.2.3.	Network Element Global Controller	5-45
5.6.6.1.2.4.	Scoreboard	5-45
5.6.6.1.2.5.	Inter-Fault Set Communication Links.....	5-45
5.6.6.1.2.6.	Voted Reset	5-45
5.6.6.1.2.7.	Fault Tolerant Clock	5-45
5.6.6.1.3.	FCR Backplane Bus Self Tests.....	5-46
5.6.6.1.4.	Input/Output Device Self Tests	5-46
5.6.6.1.5.	Power Conditioner Self Tests	5-46
5.6.6.1.6.	Mass Memory Self Tests.....	5-46
5.6.6.1.7.	System Tests.....	5-46
5.6.6.2.	Operational Constraints of Fault Detection Mechanisms.....	5-48
5.6.6.2.1.	I-BIT Mode Self Tests.....	5-50
5.6.6.2.2.	M-BIT Mode Self Tests	5-51
5.6.6.2.3.	I-BIT Mode System Tests.....	5-51
5.6.6.2.4.	M-BIT Mode System Tests	5-52
5.6.6.2.5.	C-BIT Mode Tests	5-52
5.6.6.3.	Mapping of Fault Detection Mechanisms to Test Modes	5-52
5.6.6.3.1.	Processor Self Tests	5-53
5.6.6.3.2.	Network Element Self Tests	5-55
5.6.6.3.3.	FCR Backplane Bus Self Tests.....	5-56
5.6.6.3.4.	Input/Output Device Self Tests	5-56
5.6.6.3.5.	Power Conditioner Self Tests	5-56
5.6.6.3.6.	Mass Memory Self Tests.....	5-56
5.6.6.3.7.	System Tests.....	5-57
5.6.7.	Fault Diagnosis.....	5-57
5.6.7.1.	Non-Fault Tolerant Operations	5-60
5.6.7.2.	Fault Tolerant Operations.....	5-60
5.6.7.2.1.	Local Fault Detection and Isolation	5-60
5.6.7.2.1.1.	Intra Virtual Group Presence Test.....	5-62
5.6.7.2.1.2.	Syndrome Analysis	5-62
5.6.7.2.1.3.	Self Tests.....	5-63
5.6.7.2.2.	System Fault Detection and Isolation	5-64
5.6.8.	Recovery options	5-65
5.6.8.1.	Response to Failure of Test.....	5-66
5.6.8.2.	System Recovery.....	5-67
5.6.8.2.1.	Recovery from Processor Failure	5-69
5.6.8.2.1.1.	Graceful Degradation	5-69
5.6.8.2.1.2.	Processor Resynchronization	5-70
5.6.8.2.1.3.	Processor Reintegration	5-73
5.6.8.2.1.4.	Processor Replacement.....	5-73
5.6.8.2.1.5.	Processor Replacement with Initialization.....	5-73
5.6.8.2.1.6.	Task Migration.....	5-74
5.6.8.2.2.	Recovery from Network Element Failure	5-74
5.6.8.2.2.1.	Network Element Resynchronization.....	5-75
5.6.8.2.2.2.	Network Element Masking.....	5-75
5.6.9.	Transient Fault Analysis	5-75
5.6.9.1.	Transient Recovery Option.....	5-76
5.6.9.1.1.	Processor Recovery.....	5-78
5.6.9.1.2.	Network Element Recovery.....	5-79
5.6.9.2.	Wait and See Transient Analysis Option.....	5-79
5.6.9.3.	No Transient Fault Analysis Option.....	5-80
5.6.9.4.	Hybrid Transient Fault Analysis Option.....	5-80
5.6.9.5.	Intermittent Fault Analysis	5-82

5.6.9.6.	Transient Fault Analysis Option and System Modes	5-82
5.6.10.	Fault Logging	5-82
5.6.11.	Fault Reporting	5-83
5.6.11.1.	Cockpit Display Unit	5-84
5.6.11.2.	Portable Intelligent Maintenance Aid	5-86
5.6.11.3.	Fault Annunciator Panel	5-87
5.7.	I/O Services	5-87
5.7.1.	The AFTA I/O User Interface	5-88
5.7.2.	Input/Output User View	5-88
5.7.3.	I/O Request Construction	5-89
5.7.4.	I/O Transactions	5-90
5.7.5.	I/O Chains	5-93
5.7.6.	I/O Requests	5-94
5.7.7.	I/O Data Access	5-95
5.7.8.	The Buffer Control Procedures	5-95
5.7.9.	The AFTA I/O Communication Manager	5-98
5.7.9.1.	The Nonpreemptable I/O Dispatcher	5-99
5.7.9.1.1.	The I/O Request Tasks	5-102
5.7.9.1.2.	Dispatching	5-102
5.7.9.2.	AFTA Input Output Services: Examples	5-103
5.7.9.2.1.	Example #1: All I/O Requests can be Completed in 10 ms.	5-103
5.7.9.2.2.	Example #2: All I/O Requests can not be Completed in 10 ms.	5-103
6.	Fault-Tolerant Data Bus	6-1
6.1.	Objective and Approach	6-1
6.2.	Fault-Tolerant Data Bus Requirements	6-2
6.2.1.	Packet Requirements	6-2
6.2.1.1.	Word Length	6-2
6.2.1.2.	Packet Length	6-2
6.2.2.	Network Control Requirements	6-2
6.2.2.1.	Access Control Modes	6-2
6.2.2.2.	Address Modes	6-2
6.2.2.3.	Uncontrolled Transmit Inhibit	6-3
6.2.2.4.	Flow Control	6-3
6.2.3.	Network Function Requirement	6-3
6.2.3.1.	Broadcast and Multicast Functions	6-3
6.2.3.2.	Periodic and Aperiodic Transfers	6-3
6.2.3.3.	Packet Ordering	6-4
6.2.3.4.	Station Identification	6-4
6.2.4.	Topology and Architecture Requirements	6-4
6.2.4.1.	Growth	6-4
6.2.4.2.	Topology	6-4
6.2.4.3.	Station Insertion and Removal	6-4
6.2.4.4.	Bridges for Interconnected Buses	6-4
6.2.5.	Physical Requirements	6-5
6.2.5.1.	Serial Transmission	6-5
6.2.5.2.	Media Support	6-5
6.2.5.3.	Electrical Isolation	6-5
6.2.5.4.	Station Separation	6-5
6.2.6.	Fault Tolerance Requirements	6-5
6.2.6.1.	Packet Delivery	6-5
6.2.6.2.	Synchronization	6-5
6.2.6.3.	Source Congruency	6-6

6.2.6.4.	Connectivity.....	6-6
6.2.6.5.	Station to Network Interface.....	6-6
6.2.6.6.	Redundancy.....	6-6
6.2.6.7.	Station Redundancy.....	6-6
6.2.6.8.	Error Detection.....	6-6
6.2.6.9.	Diagnosability.....	6-7
6.2.6.10.	Self-Test.....	6-7
6.2.6.11.	Byzantine Resilience.....	6-7
6.2.6.12.	Fault Isolation and Containment.....	6-7
6.2.7.	Performance Requirements.....	6-7
6.2.7.1.	Message Priorities.....	6-8
6.2.7.2.	Network Bandwidth.....	6-8
6.2.7.3.	Initialization Time.....	6-8
6.3.	FTDB Architecture Study.....	6-8
6.3.1.	Broadcast Buses.....	6-11
6.3.2.	Token Rings.....	6-13
6.3.3.	Circuit Switched Network.....	6-15
6.3.4.	Packet Switched Network.....	6-16
6.3.5.	Fiber Optic Networks.....	6-16
6.3.6.	Authentication Protocols.....	6-20
6.4.	Existing and Proposed Standards.....	6-20
6.4.1.	AIPS Intercomputer Network.....	6-21
6.4.2.	SAVA High-Speed Data Bus.....	6-22
6.4.3.	JIAWG High-Speed Data Bus.....	6-23
6.4.4.	Fiber Distributed Data Interface (FDDI).....	6-24
6.4.5.	SAFENET II.....	6-25
6.4.6.	Summary.....	6-26
6.5.	FTDB Brassboard Design Proposal.....	6-34
6.5.1.	Physical Layer.....	6-34
6.5.1.1.	Physical Layer Protocol.....	6-35
6.5.1.2.	Physical Layer Medium Dependent.....	6-35
6.5.2.	Data Link Layer.....	6-36
6.5.2.1.	Media Access Control.....	6-37
6.5.2.2.	Station Management.....	6-37
6.5.2.3.	Logical Link Control.....	6-37
6.5.3.	Network Layer.....	6-38
6.5.3.1.	Byzantine Resilient Network Protocol (BRNP).....	6-40
6.5.3.2.	Authentication Protocol (ATP).....	6-41
6.5.3.3.	Address Resolution Protocol (ARP).....	6-41
6.5.4.	Transport Layer.....	6-42
6.5.4.1.	Periodic Datagram Protocol (PDP).....	6-42
6.5.4.2.	Services Transaction Protocol (STP).....	6-42
6.5.4.3.	Asynchronous Datagram Protocol (ADP).....	6-43
6.5.4.4.	Network Data Stream Protocol (NDSP).....	6-43
6.5.4.5.	Network Diagnostic Protocol (NDP).....	6-43
6.5.4.6.	Echo Protocol (EP).....	6-44
6.5.4.7.	Time Management Protocol (TMP).....	6-44
6.6.	FTDB Development Plan.....	6-44
6.6.1.	Developmental and Non-developmental Items.....	6-45
6.6.2.	Proposed FTDB Brassboard Development Plan.....	6-45
6.6.2.1.	Subtask 1-Authentication Protocols.....	6-45
6.6.2.2.	Subtask 2-Byzantine Resilience.....	6-46
6.6.2.3.	Subtask 3-Network FDIR.....	6-46
6.6.2.4.	Subtask 4-Transport Layer Protocols.....	6-46

6.6.3.	FTDB Brassboard Development Schedule.....	6-47
7.	Testability and Maintainability.....	7-1
7.1.	Level of testing.....	7-1
7.1.1.	Component self tests	7-1
7.1.2.	System tests	7-2
7.2.	Test Modes	7-2
7.3.	Operator interface.....	7-6
7.4.	FTPP C2 Network Element Tests	7-6
7.4.1.	Off-line Standalone NE Diagnostic Tests	7-8
7.4.2.	Functional Block: Processor-Network Element Interface.....	7-9
7.4.3.	Functional Block: Network Element Data Paths	7-11
7.4.4.	Functional Block: Network Element Global Controller	7-13
7.4.5.	Functional Block: The Scoreboard.....	7-14
7.4.6.	Functional Block: The Inter-Fault Set Communication Links	7-15
7.4.7.	Conclusions	7-15
7.5.	AFTA Maintenance.....	7-16
7.6.	AFTA Line Maintenance Procedure	7-16
8.	Common Mode Fault Study	8-1
8.1.	Objective.....	8-1
8.2.	Approach	8-1
8.3.	Enumeration of Common Mode Fault Sources	8-2
8.3.1.	Classification by Nature.....	8-2
8.3.2.	Classification by Origin	8-2
8.3.3.	Classification by persistence.....	8-3
8.4.	Enumeration of Common Mode Fault Avoidance, Removal, Tolerance Techniques	8-4
8.4.1.	Common Mode Fault Avoidance.....	8-5
8.4.1.1.	Formal Methods	8-5
8.4.1.2.	Formally Verified Components.....	8-5
8.4.1.3.	Mature Components.....	8-5
8.4.1.4.	Design Automation Tools	8-6
8.4.1.5.	Architectural Considerations	8-7
8.4.1.6.	Design Diversity.....	8-8
8.4.1.7.	Use of Standards	8-8
8.4.1.8.	Good Software Engineering Practices	8-9
8.4.1.9.	Conservative Hardware Design Practices	8-9
8.4.1.10.	Shielding, Packaging and Thermal Management	8-11
8.4.2.	Common Mode Fault Removal.....	8-11
8.4.2.1.	Design Reviews	8-11
8.4.2.2.	Simulations.....	8-11
8.4.2.3.	Testing.....	8-11
8.4.2.4.	Fault Injection.....	8-12
8.4.2.5.	Discrepancy Reports	8-13
8.4.2.6.	Automated Theorem Provers.....	8-16
8.4.3.	Common Mode Fault Tolerance.....	8-16
8.4.3.1.	Common Mode Fault Detection.....	8-16
8.4.3.2.	Common Mode Fault Recovery	8-18
8.4.3.3.	Performance Overheads of Common Mode Fault Tolerance Techniques	8-19
8.4.3.4.	Common Mode Fault Examples	8-19
8.5.	Effectiveness of Common Mode Fault Avoidance, Fault Removal, Fault Tolerance Techniques	8-22
8.6.	Suitability of Common Mode Fault Avoidance, Fault Removal, Fault Tolerance Techniques for AFTA.....	8-24

8.7.	Plan for Implementation of CMF Avoidance, Removal, Tolerance Techniques	8-30
8.7.1.	Demonstration/Validation Phase	8-30
8.7.1.1.	AFTA System.....	8-31
8.7.1.2.	Hardware Design.....	8-31
8.7.1.3.	Software Design.....	8-32
8.7.1.4.	Hardware-Software Test and Integration.....	8-33
8.7.2.	Full Scale Development Phase	8-34
8.7.3.	Production Phase	8-34
8.7.4.	Deployment Phase	8-35
8.7.5.	Pre-Planned Product Improvement Phase	9-1
9.	Analytical Models	9-2
9.1.	Performance Model.....	9-2
9.1.1.	Delivered Throughput	9-7
9.1.2.	Intertask Communication	9-10
9.1.3.	Input/Output	9-11
9.2.	Reliability and Availability Models	9-18
9.2.1.	Formulation for Graceful Degradation Class of Fault Recovery	9-20
9.2.2.	Formulation for Processor Replacement Class of Fault Recovery	9-22
9.2.3.	Failure Rate Calculation Methodology	9-22
9.2.3.1.	Environmental Effects	9-23
9.2.3.2.	PE Failure Rate Calculations.....	9-24
9.2.3.2.1.	Hiatus: Ground Fixed	9-24
9.2.3.2.2.	Aircraft Mission	9-24
9.2.3.2.3.	Ground Mission.....	9-25
9.2.3.3.	NE Failure Rate Calculations	9-25
9.2.3.3.1.	Methodology.....	9-25
9.2.3.3.2.	Assumptions.....	9-28
9.2.3.3.3.	AFTA Hiatus NE Failure Rate.....	9-30
9.2.3.3.4.	AFTA Aircraft Mission NE Failure Rate	9-32
9.2.3.3.5.	AFTA Ground Mission NE Failure Rate.....	9-33
9.2.3.3.6.	Implications and Indicated Course of Action	9-37
9.2.3.4.	IOC Failure Rate Calculations	9-38
9.2.3.5.	PC Failure Rate Calculations.....	9-38
9.2.3.5.1.	Hiatus.....	9-38
9.2.3.5.2.	Aircraft Mission	9-39
9.2.3.5.3.	Ground Mission.....	9-39
9.3.	Physical Characteristics (WPV) Models	9-39
9.3.1.	Weight.....	9-39
9.3.2.	Power	9-40
9.3.3.	Volume.....	9-40
9.4.	Fleet Life-Cycle Cost per Service Unit (FLCCPSU) Model	9-41
9.4.1.	Assumptions and Analysis Inputs	9-43
9.4.2.	Application Scenario.....	9-43
9.4.3.	Procurement Cost.....	9-44
9.4.4.	Manpower Cost due to Repairs	9-45
9.4.5.	Cost due to Spares.....	9-46
9.4.6.	Cost due to Unreliability of AFTA.....	9-46
9.4.7.	Total FLCCPSU	10-1
10.	VHSIC Hardware Description Language	10-1
10.1.	VHDL Overview.....	10-2
10.1.1.	Behavioral vs. Structural Models	10-4
10.1.2.	Overview of a VHDL Description	10-5
10.2.	Use of VHDL for AFTA.....	

10.2.1.	Design.....	10-6
10.2.1.1.	Behavioral.....	10-6
10.2.1.2.	Structural.....	10-7
10.2.2.	Simulation.....	10-7
10.2.3.	Testing.....	10-8
10.2.4.	Documentation.....	10-8
10.2.4.1.	Custom Devices.....	10-8
10.2.4.2.	Standard Devices.....	10-10
10.2.5.	Candidate VHDL Tools for AFTA NE Design.....	10-12
10.2.6.	Compliance with Data Item Description.....	10-12
10.2.6.1.	Reference Documents.....	10-12
10.2.6.2.	VHDL Model Hierarchy.....	10-12
10.2.6.3.	Leaf-Level Modules.....	10-12
10.2.6.4.	Entity Declarations.....	10-13
10.2.6.5.	Behavioral Body.....	10-14
10.2.6.6.	Structural Body.....	10-14
10.2.6.7.	VHDL Simulation Support.....	10-14
10.2.6.8.	Error Messages.....	10-15
10.2.6.9.	Annotations.....	10-15
10.2.6.10.	Reference to Origin.....	10-15
10.2.6.11.	VHDL Documentation Format.....	10-15
11.	AFTA Validation and Verification.....	11-1
11.1.	Verifiable AFTA Attributes.....	11-4
11.2.	Verification of Byzantine Resilience and Operational Correctness.....	11-5
11.2.1.	Fault Containment.....	11-6
11.2.2.	NE Synchronization.....	11-6
11.2.3.	Interactive Consistency.....	11-6
11.2.4.	Voting.....	11-7
11.2.5.	Message-Release Authorization (Scoreboard).....	11-7
11.2.6.	Reconfigurability.....	11-8
11.2.7.	Functional Synchronization.....	11-9
11.2.8.	Byzantine Resilient Virtual Circuit Abstraction.....	11-9
11.2.9.	Rate Group Scheduling.....	11-10
11.2.10.	Intertask Communication Services.....	11-12
11.2.11.	I/O System Services.....	11-12
11.2.12.	Redundancy Management (FDIR) Software.....	11-13
11.3.	Verification of Performance Predictions.....	11-14
11.3.1.	Delivered Throughput per VG.....	11-14
11.3.2.	Available Memory per VG.....	11-16
11.3.3.	Effective Intertask Communication Bandwidth and Latency.....	11-16
11.3.4.	Effective I/O Bandwidth and Latency.....	11-16
11.3.5.	Iteration Rate of a Task.....	11-18
11.4.	Verification of Reliability and Availability Predictions.....	11-18
11.4.1.	Component Failure Rate.....	11-19
11.4.2.	Fault Reconfiguration Time.....	11-20
11.4.3.	Fault Reconfiguration Coverage.....	11-21
11.4.4.	VG Redundancy Levels.....	11-22
11.4.5.	Mission/Hiatus Time.....	11-22
11.5.	Verification of Cost Predictions.....	11-22
11.6.	Verification of Weight, Power, and Volume Predictions.....	11-23
12.	AFTA Architecture Synthesis.....	12-1
12.1.	AFTA Architecture Synthesis.....	12-1
12.1.1.	Configurable Parameters.....	12-1
12.1.2.	AFTA Architecture Synthesis Procedure.....	12-2

12.2. AFTA Architecture Synthesis	12-3
12.2.1. AFTA Characteristics Common to Both Missions	12-3
12.2.1.1. Delivered Throughput	12-3
12.2.2. AFTA Configuration for TF/TA/NOE/FCS Mission	12-4
12.2.2.1. Analytical Results	12-5
12.2.2.1.1. Failure Rates	12-5
12.2.2.1.2. Reliability	12-5
12.2.2.1.3. Throughput-Reliability Tradeoff	12-8
12.2.2.1.4. Effect of VHSIC/VLSI Network Element Technology on Reliability	12-9
12.2.2.1.5. Unavailability	12-10
12.2.2.1.6. Weight	12-12
12.2.2.1.7. Power	12-13
12.2.2.1.8. Volume	12-14
12.2.2.1.9. Cost	12-15
12.2.3. AFTA Analysis for Ground Vehicle Mission	12-20
12.2.3.1. Throughput	12-20
12.2.3.2. Reliability	12-20
12.2.3.3. Weight	12-24
12.2.3.4. Power	12-25
12.2.3.5. Volume	12-26
Appendix A. References	A-1
Appendix B. Glossary of Terms and Acronyms	B-1

This page intentionally left blank.

List of Figures

Figure 4-1.	AFTA Physical Configuration	4-2
Figure 4-2.	AFTA Virtual Configuration	4-3
Figure 4-3.	Network Element Addresses	4-7
Figure 4-4.	Absolute Mask	4-8
Figure 4-5.	Relative Mask	4-8
Figure 4-6.	ISYNC Procedure	4-14
Figure 4-7.	NE Memory Map	4-21
Figure 4-8.	DPRAM Memory Map	4-22
Figure 4-9.	Packet Class Field	4-24
Figure 4-10.	Vote Error Field	4-26
Figure 4-11.	Clock Error Field	4-27
Figure 4-12.	Link Error Field	4-27
Figure 4-13.	OBNE Timeout Field	4-28
Figure 4-14.	IBNF Timeout Field	4-28
Figure 4-15.	Scoreboard Vote Error Field	4-28
Figure 4-16.	Buffer Manager Memory Map	4-30
Figure 4-17.	Next Port Format	4-30
Figure 4-18.	Ready Port Format	4-31
Figure 4-19.	CT Update Packet Format	4-32
Figure 4-20.	Redundancy Level Field	4-32
Figure 4-21.	PE Mask Field	4-32
Figure 4-22.	Processor Specification Field	4-33
Figure 4-23.	NE Mask Format	4-33
Figure 4-24.	TNR Receive Packet Format	4-34
Figure 4-25.	TNR Result Byte Format	4-35
Figure 4-26.	VRESET Packet	4-35
Figure 4-27.	VRESET Command Byte	4-36
Figure 4-28.	AFTA FCR Architecture	4-37
Figure 4-29.	SAVA-based AFTA FCR	4-38
Figure 4-30.	SAVA-based AFTA LRM	4-39
Figure 4-31.	JIAWG-based AFTA FCR	4-40
Figure 4-32.	JIAWG-based AFTA LRM	4-41
Figure 4-33.	Functional Block Diagram of an AFTA Processing Element	4-42
Figure 4-34.	Functional Block Diagram of the AFTA Network Element	4-48
Figure 4-35.	Inter-FCR Fiber-Optic Network	4-51
Figure 4-36.	Normal FTC Adjustment Period	4-52
Figure 4-37.	Self-Ahead FTC Adjustment Period	4-53
Figure 4-38.	Self-Behind FTC Adjustment Period	4-53
Figure 4-39.	Network Element Brassboard Layout	4-56
Figure 4-40.	Functional Block Diagram of the AFTA Power Conditioner	4-69
Figure 5-1.	AFTA System Software Organization	5-1
Figure 5-2.	Example AFTA Configuration	5-3
Figure 5-3.	Rate Group Frame Phasing	5-5
Figure 5-4.	VG Configuration Table	5-6
Figure 5-5.	Task Configuration Table	5-8
Figure 5-6.	Task Configuration Table Initialization	5-9
Figure 5-7.	Rate Group Task Lists	5-11
Figure 5-8.	Initialize Rate Group Tasking Procedure	5-12
Figure 5-9.	Task Priority	5-13

Figure 5-10.	Mapping of RG Frames to Minor Frames	5-14
Figure 5-11.	Frame Start Procedure	5-15
Figure 5-12.	Example Rate Group Task.....	5-17
Figure 5-13.	Wait for Next Frame Procedure.....	5-17
Figure 5-14.	Initialize Time Keeping Procedure.....	5-19
Figure 5-15.	Task-to-Task Message and Packet Formats	5-23
Figure 5-16.	Transmit and Receive Queues.....	5-25
Figure 5-17.	Transmit and Receive Packet Queue Entries.....	5-27
Figure 5-18.	Message Class and Message Data Structure.....	5-28
Figure 5-19.	CID Queue Table	5-29
Figure 5-20.	Initialize Communication Procedure.....	5-29
Figure 5-21.	Send Message Procedure	5-30
Figure 5-22.	Queue Message Procedure.....	5-31
Figure 5-23.	Send Queue Procedure.....	5-32
Figure 5-24.	Message Pending Table.....	5-33
Figure 5-25.	CID Status Table.....	5-34
Figure 5-26.	Read Message Procedure	5-35
Figure 5-27.	Update Frame Marker Procedure	5-36
Figure 5-28.	Retrieve Message Procedure	5-36
Figure 5-29.	System mode and test mode interactions.....	5-38
Figure 5-30.	Operational modes.....	5-40
Figure 5-31.	Test Mode Sequences	5-50
Figure 5-32.	Off-Line FDI Overview.....	5-59
Figure 5-33.	Synchronous FDI Overview	5-61
Figure 5-34.	Qualitative evaluation of recovery methods.....	5-68
Figure 5-35.	Lost Channel Synchronization	5-72
Figure 5-36.	Transient Recovery Algorithm	5-77
Figure 5-37.	Wait and See Transient Fault Analysis algorithm.....	5-79
Figure 5-38.	No Transient Fault Analysis algorithm.....	5-80
Figure 5-39.	Hybrid Transient Fault Analysis algorithm.....	5-81
Figure 5-40.	Possible Mapping of Transient Analysis Options to System Modes.....	5-82
Figure 5-41.	AFTA System Level Display	5-84
Figure 5-42.	LRU Level Display.....	5-85
Figure 5-43.	LRM Level Display.....	5-86
Figure 5-44.	The AFTA I/O Services.....	5-87
Figure 5-45.	The I/O User Interface and I/O Communication Manager	5-88
Figure 5-46.	I/O Transactions, I/O Chains, and I/O Requests.....	5-91
Figure 5-47.	An Input Transaction Record.....	5-92
Figure 5-48.	An Output Transaction Record.....	5-92
Figure 5-49.	An Input/Output Transaction Record.....	5-92
Figure 5-50.	The Create_Transaction Procedure	5-93
Figure 5-51.	The Create_Chain Procedure.....	5-93
Figure 5-52.	The Create_I/O Request Procedure.....	5-94
Figure 5-53.	The Lock_I/O_Request_Buffer Procedure	5-96
Figure 5-54.	The Lock_I/O_Request_Buffer Procedure	5-96
Figure 5-55.	The I/O_Buffer_Contention Exception and Error Handler	5-97
Figure 5-56.	The I/O User Interface Status Retrieval Procedures	5-97
Figure 5-57.	The I/O Communication Manager	5-99
Figure 5-58.	The Nonpreemptable I/O Dispatcher	5-101
Figure 5-59.	I/O Requests for Example #1.....	5-105
Figure 5-60.	Example #1.....	5-105
Figure 5-61.	I/O Requests for Example #2.....	5-106
Figure 5-62.	Example #2.....	5-106

Figure 6-1.	Broadcast Bus Topology	6-9
Figure 6-2.	Token Ring Topology.....	6-11
Figure 6-3.	Fully Braided Chordal Ring.....	6-12
Figure 6-4.	Dual Counter-rotating Ring.....	6-13
Figure 6-5.	Example Circuit Switched Topology.....	6-14
Figure 6-6.	AIPS IC Network	6-21
Figure 6-7.	ISO/OSI Model of FTDB	6-27
Figure 6-8.	FTDB Architecture	6-30
Figure 6-9.	Triply Redundant VG T1 Simultaneously Writes Output Data to Signer/Checker Components of Fault Tolerant Data Bus.....	6-31
Figure 6-10.	Step 1 of FTDB Message Transfer.....	6-32
Figure 6-11.	Step 2 of FTDB Message Transfer.....	6-32
Figure 6-12.	Step 3 of FTDB Message Transfer.....	6-33
Figure 6-13.	Step 4 of FTDB Message Transfer.....	6-33
Figure 6-14.	Step 5 of FTDB Message Transfer.....	6-34
Figure 6-15.	FTDB Brassboard Development Schedule	6-47
Figure 7-1.	System mode and Test Mode Interactions	7-4
Figure 7-2.	Test Mode State Diagram	7-5
Figure 7-3.	Block Diagram of FTTP C2 Network Element	7-7
Figure 7-4.	Maintenance-Related AFTA Features	7-17
Figure 7-5.	AFTA LRU.....	7-19
Figure 8-1.	Typical Discrepancy Report Format.....	8-15
Figure 9-1.	AFTA Methodology Information Flow	9-1
Figure 9-2.	Mapping of RG Frames to Minor Frames	9-3
Figure 9-3.	RG Message Passing.....	9-8
Figure 9-4.	Outgoing Message Processing	9-9
Figure 9-5.	Graceful Degradation of Quadruplex VG1	9-13
Figure 9-6.	Processor Replacement Redundancy Management for Quadruplex VG1	9-15
Figure 9-7.	AFTA Hiatus NE Failure Rate Constituents.....	9-30
Figure 9-8.	AFTA Flight Mission NE Failure Rate Constituents.....	9-31
Figure 9-9.	AFTA Ground Mission NE Failure Rate Constituents.....	9-32
Figure 9-10.	Helicopter TF/TA/NOE Mission Scenario State Diagram.....	9-41
Figure 10-1.	Hierarchical VHDL Model.....	10-2
Figure 10-2.	Behavioral Model of a D Flip-Flop.....	10-3
Figure 10-3.	Structural Model of a D Flip-Flop	10-3
Figure 10-4.	Behavioral Model of a NAND Gate	10-3
Figure 10-5.	Reprocurement Options for Custom Devices.....	10-9
Figure 10-6.	Reprocurement Options for Standard Devices	10-11
Figure 12-1.	AFTA Delivered Throughput vs. Number of Processing Elements	12-4
Figure 12-2.	Probability of AFTA Failure for 1-hour Rotary Wing Aircraft Mission	12-6
Figure 12-3.	Probability of AFTA Failure for 4-hour Rotary Wing Aircraft Mission	12-7
Figure 12-4.	Delivered Throughput vs. AFTA Failure Probability for 1-hour Rotary Wing Aircraft Mission.....	12-8
Figure 12-5.	Probability of AFTA Failure for 1-hour Rotary Wing Aircraft Mission using VHSIC/VLSI-based Network Element.....	12-9
Figure 12-6.	Ratio of Probability of AFTA Failure for 1-hour Rotary Wing Aircraft Mission.....	12-10
Figure 12-7.	AFTA Unavailability after 23-hour Hiatus for Six-VG/Four-FCR MDC	12-11
Figure 12-8.	AFTA Unavailability after 23-hour Hiatus for Six-VG/Five-FCR MDC	12-12

Figure 12-9.	AFTA Weight for Helicopter Mission.....	12-13
Figure 12-10.	AFTA Power for Helicopter Mission	12-14
Figure 12-11.	AFTA Volume for Helicopter Mission.....	12-15
Figure 12-12.	AFTA Unreliability for Eight-Hour Ground Vehicle Mission.....	12-21
Figure 12-13.	AFTA Unreliability for 24-Hour Ground Vehicle Mission.....	12-22
Figure 12-14.	AFTA Unreliability for 168-Hour Ground Vehicle Mission	12-23
Figure 12-15.	AFTA Unreliability for 720-Hour Ground Vehicle Mission	12-24
Figure 12-16.	AFTA Weight for Ground Vehicle Mission	12-25
Figure 12-17.	AFTA Power for Ground Vehicle Mission.....	12-26
Figure 12-18.	AFTA Volume for Ground Vehicle Mission	12-27

List of Tables

Table 4-1.	Characteristics of Radstone PMV 68M CPU-3A Processing Element	4-44
Table 4-2.	Characteristics of Lockheed Sanders STAR MVP Processing Element	4-45
Table 4-3.	Characteristics of SAVA GPPM Processing Element.....	4-46
Table 4-4.	Characteristics of AFTA Network Element.....	4-55
Table 4-5.	Military Device Availability for Network Element.....	4-58
Table 6-1.	Comparison of Standards.....	6-26
Table 7-1.	AFTA Maintenance Time Line	7-16
Table 8-1.	Classification of Common Mode Faults	8-3
Table 8-2.	Commonly Observed Error Symptoms of Common Mode Faults.....	8-21
Table 8-2.	Commonly Observed Error Symptoms of Common Mode Faults (Cont.)	8-22
Table 8-3.	Effectiveness of CMF A/R/T Techniques.....	8-23
Table 8-4.	Application of CMF A/R/T Techniques to AFTA.....	8-29
Table 9-1.	Completed/Started RGs vs. Frame Boundary.....	9-4
Table 9-2.	Environmental Failure Rate Multipliers for Monolithic Microelectronic Devices.....	9-23
Table 9-3.	PE Cited Failure Rate Data	9-23
Table 9-4.	PE Hiatus Failure Rate Data	9-24
Table 9-5.	PE Aircraft Mission Failure Rate Data	9-24
Table 9-6.	PE Aircraft Mission Failure Rate Data	9-24
Table 9-7.	AFTA Baseline NE Parts Hiatus Failure Rates	9-29
Table 9-8.	AFTA Baseline NE Hiatus Failure Rate and Constituents.....	9-30
Table 9-9.	AFTA Baseline NE Flight Mission Failure Rate and Constituents	9-31
Table 9-10.	AFTA Baseline NE Ground Mission Failure Rate and Constituents	9-32
Table 9-11.	Summary of AFTA Baseline NE MTBF Data.....	9-33
Table 9-12.	Gates, Pins, and Power Consumption of High-End NE	9-33
Table 9-13.	Comparison of MTBFs of Baseline and High End AFTA Network Element	9-37
Table 9-14.	PC Cited Failure Rate Data	9-38
Table 11-1.	Verifiable AFTA Attributes.....	11-4
Table 11-2.	Completed/Started RGs vs. Minor Frame Boundary.....	11-11
Table 11-3.	Verification of Delivered Throughput	11-15
Table 11-4.	Verification of Intertask Communication Bandwidth and Latency.....	11-17
Table 11-5.	Verification of I/O Communication Bandwidth and Latency	11-17
Table 11-6.	VG Failure Probability Due to Attrition and Near-Coincident Faults.....	11-21
Table 12-1.	AFTA Component Failure Rates for Helicopter Mission Scenario	12-5
Table 12-2.	AFTA Component Weights for Helicopter AFTA.....	12-13
Table 12-3.	AFTA Component Powers for Helicopter AFTA.....	12-14
Table 12-4.	AFTA Component Volumes for Helicopter AFTA	12-15
Table 12-5.	FLCCPSU Input Parameters for Rotary Wing Aircraft Mission.....	12-16
Table 12-6.	FLCCPSU Output Parameters for Rotary Wing Aircraft Mission, Four-FCR MDC with no spare FCR.....	12-17
Table 12-7.	FLCCPSU Constituent Costs for Four-FCR MDC with no spare FCR	12-18

Table 12-8.	FLCCPSU Output Parameters for Rotary Wing Aircraft Mission– Four-FCR MDC with one spare FCR.....	12-18
Table 12-9.	FLCCPSU Constituent Costs for Four-FCR MDC with one spare FCR	12-18
Table 12-10.	FLCCPSU Output Parameters for Rotary Wing Aircraft Mission– Five-FCR MDC.....	12-19
Table 12-11.	Minimum-FLCCPSU Constituent Costs for Four-FCR MDC with one spare FCR	12-19
Table 12-12.	AFTA Component Failure Rates for Ground Vehicle Mission Scenario.....	12-21
Table 12-13.	AFTA Component Weights for Helicopter AFTA	12-24
Table 12-14.	AFTA Component Powers for Ground Vehicle AFTA	12-25
Table 12-15.	AFTA Component Volumes for Ground Vehicle AFTA.....	12-26

Introduction to Volumes I and II

The long-term objective of the AFTA program is to develop and deploy the Army Fault Tolerant Architecture (AFTA) on a variety of Army programs such as the Computer-Aided Low Altitude Helicopter Flight Program and the Armored Systems Modernization (ASM) vehicles. Applications such as these may be characterized by a combination of computational intensiveness, real-time response requirements, high reliability and availability requirements, and maintainability, testability, and producibility requirements.

The AFTA architecture is based on the Charles Stark Draper Laboratory, Inc. Fault Tolerant Parallel Processor (FTPP). AFTA is a real-time computer possessing high reliability, maintainability, availability, testability, and computational capability. It achieves the first four properties primarily through adherence to a theoretically rigorous theory of fault tolerance known as Byzantine Resilience, through which arbitrary failure modes can be tolerated. It is designed for verifiability and quantifiability of key system attributes with a high degree of confidence, in part due to its theoretically sound basis and in part due to plausible parameterizations of fault tolerance and Operating System overheads. Through the use of parallel processing, AFTA achieves sufficient throughput for future integrated avionics and control functions. To be useful for a variety of Army applications, the number and redundancy level of processing sites in AFTA may be varied from one application to another, and AFTA is programmed in the DoD-mandated Ada language. AFTA is intended to be relatively easy to produce and upgrade through extensive use of Non Developmental Items and compliance with well-accepted electrical, mechanical, and functional standards.

Over the past few years NASA and the Strategic Defense Initiative Office (SDIO) have sponsored the Advanced Information Processing System (AIPS) program at Draper Laboratory. The overall goal of the AIPS program is to produce the knowledgebase necessary to achieve validated distributed fault tolerant computer system architectures for advanced real-time aerospace applications [Har91b]. As a part of this effort, an AIPS engineering model consisting of hardware building blocks such as Fault Tolerant Processors and Inter-Computer (IC) and Input/Output (I/O) networks and software building blocks such as Local System Services, IC and I/O Communications Services was constructed. AFTA can be considered to be a high-throughput AIPS building block which can be interfaced to the AIPS IC network. Section 3.7 describes the AIPS engineering model in more detail and illustrates how it can be interfaced with AFTA.

This report describes the results of the Conceptual Study phase of the AFTA development, and consists of fourteen sections in two volumes. Volume I is introductory in nature and contains Sections 1 through 3. Section 1 introduces the AFTA program, its objectives, and key elements of its technical approach. Section 2 defines a format for representing mission requirements in a manner suitable for first-order AFTA sizing and analysis, followed by a discussion of the current state of mission requirements acquisition for the targeted Army missions. Section 3 presents an overview of AFTA's architectural theory of operation.

Volume II contains detailed technical information and analyses in Sections 4 through 14. Section 4 describes the AFTA hardware architecture and components, and Section 5 describes the architecture of the AFTA Operating System. Section 6 describes the architecture and operational theory of the AFTA Fault Tolerant Data Bus. Section 7 presents the test and maintenance strategy developed for use in fielded AFTA installations. Section 8 describes an approach to be used in reducing the probability of AFTA failure due to common-mode faults. Section 9 develops analytical models for AFTA performance, reliability, availability, life cycle cost, weight, power, and volume. Section 10 presents the approach for using VHDL to describe and design AFTA's developmental hardware. Section 11 describes a plan for verifying and validating key AFTA concepts during the Dem/Val phase, and Section 12 utilizes the analytical models and partial mission requirements to generate AFTA configurations for the TF/TA/NOE and Ground Vehicle missions. References are contained in Section 13, and a glossary of terms and acronyms is included in Section 14.

Because some readers may wish only to read individual volumes, Volumes I and II contain some redundant information.

4. AFTA Hardware Architecture

Section 4 describes the hardware architecture of the AFTA. The AFTA architecture consists of a cluster of processing sites interconnected by a fault-tolerant network system constructed from custom-built Network Elements and fiber optic interconnect. The cluster also contains controllers for communication between the AFTA, other computer systems, and I/O devices.

4.1. AFTA Physical Configuration

A diagram of the physical AFTA configuration is shown in Figure 4-1. The AFTA consists of 4 or 5 fault-containment regions (FCR). Each FCR contains a Network Element (NE), 0 to 8 Processing Elements (PE), and 0 or more I/O controllers (IOC).

The Network Elements provide communication between PEs, keep the FCRs synchronized, and maintain data consensus among FCRs. The NE is designed to implement the requirements for Byzantine resilience [LSP82].

The Processing Elements are the computational sites. Each PE consists of a microprocessor, private RAM and ROM, and miscellaneous support devices, such as periodic timer interrupts. The PEs may optionally have private I/O devices, such as ethernet, RS-232, etc. The microprocessor may be either a general-purpose processor or a special-purpose processor for signal or image processing.

The I/O controllers connect the AFTA to the outside world. These I/O devices can be anything that is compatible with the bus connecting the elements within the FCR. I/O controllers may have a programmable processor on board which actually drives the I/O. These devices are referred to as smart I/O. Other I/O controllers may require an off-board processor to act as the controlling processor over the bus. These devices are referred to as dumb I/O. Smart I/O can exist in the virtual AFTA configuration as a simplex virtual group. Dumb I/O must be controlled by another processor, which could be either a simplex group or a single member of a fault-masking group. Redundant I/O (such as a dual redundant interface bus) is treated as multiple simplex devices by the AFTA.

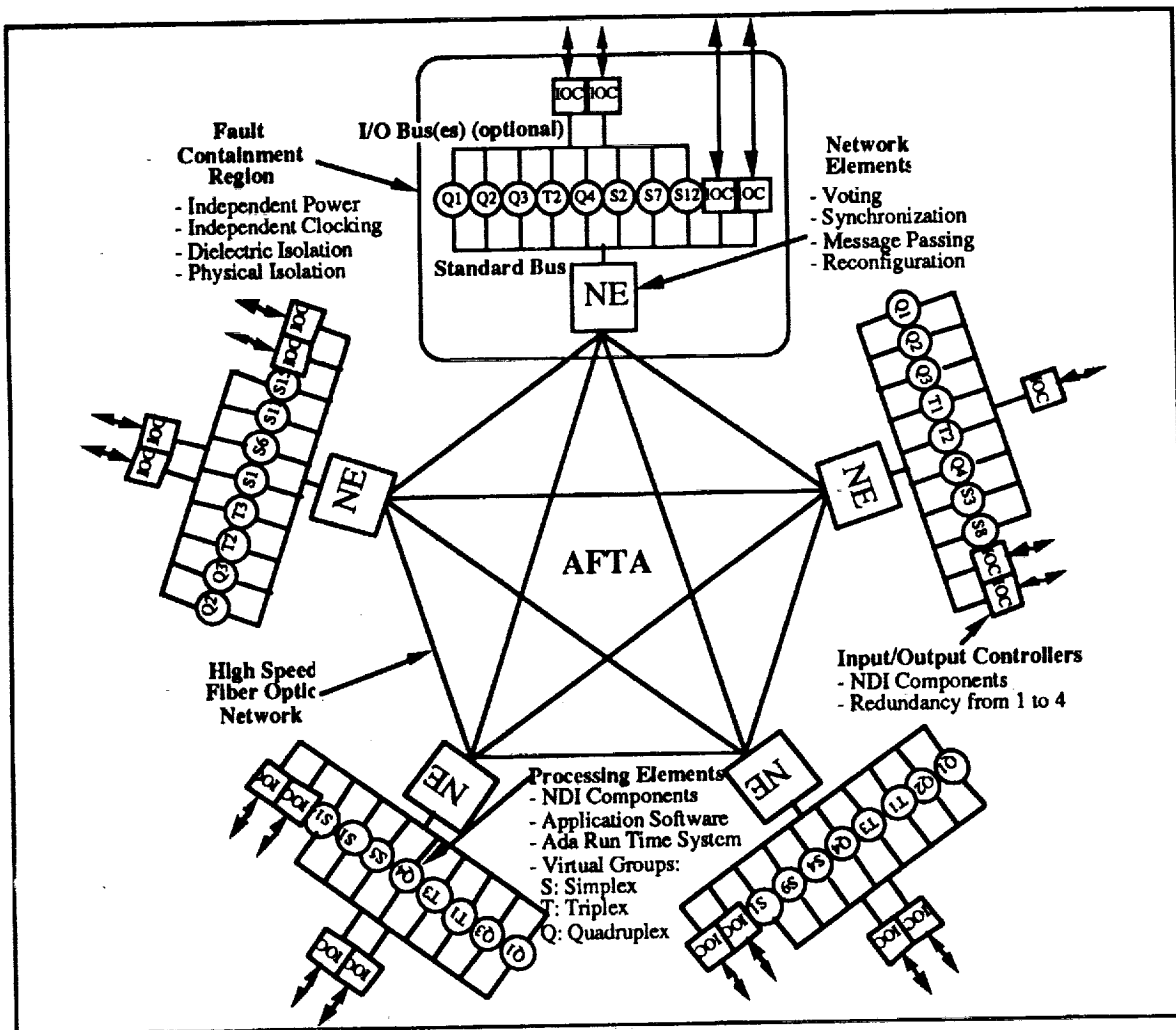


Figure 4-1. AFTA Physical Configuration

The devices in an FCR are interconnected using one or more backplane buses. PEs communicate with the NEs and IOCs through the bus(es). Data communication is usually between a PE and the NE, or between a PE and an IOC. Normally, direct PE to PE communication should not be used. If a PE wants to communicate with another PE, the exchange primitives provided by the NE should be used.

4.2. AFTA Virtual Configuration

A parallel processor is usually characterized by a network that provides interconnection between multiple processing sites. Data is passed between processing sites using a message-passing paradigm. In the AFTA, the ensemble of Network Elements provides a virtual bus topology connecting the processing sites.

The AFTA has the capability of grouping processors on the virtual bus into virtual groups. Members of a virtual group execute the same code on the same data set. These members can compare results, using the Network Elements, to mask a failure in any one of the FCRs.

The virtual bus topology of the AFTA is shown in Figure 4-2. This figure shows several example virtual groups. Virtual groups consisting of only one processing site are called simplex. These groups are not fault-masking, since there is only a single member and it is not possible to determine the validity of a single piece of data without prior knowledge of the proper value. The other types of virtual groups are triplexes and quadruplexes, consisting of three and four processing sites, respectively. These groups are called fault-masking groups (FMG), since a fault in any single member will be detected and masked by the other members.

Input and output controllers are also considered in the virtual configuration. I/O devices are assigned, either statically or dynamically, to a specific virtual group. The virtual group to which an I/O device is assigned is responsible for executing the device driver code to communicate with the device. There are three basic types of I/O configurations in the AFTA: simplex I/O assigned to a non-FMG, simplex I/O assigned to an FMG, and redundant I/O assigned to an FMG. Each of these configurations is shown in Figure 4-2. See Section 5.7 for a more complete discussion of I/O drivers in the AFTA system.

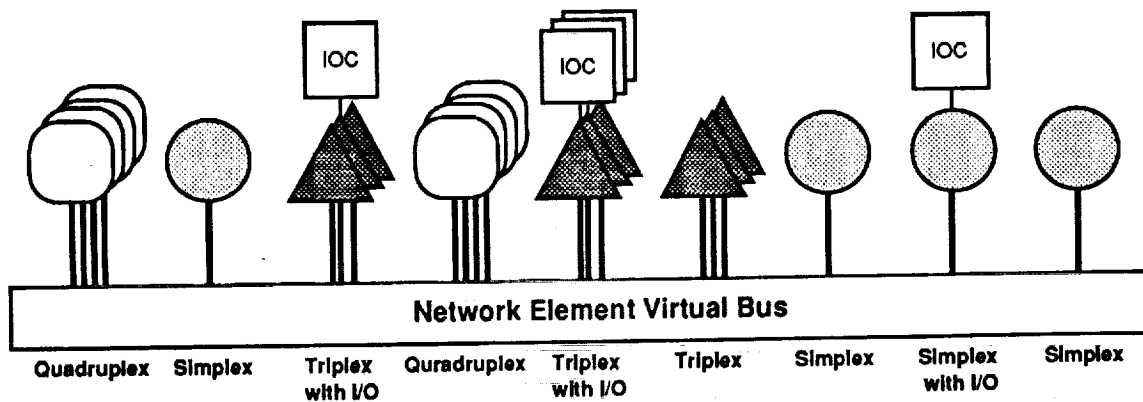


Figure 4-2. AFTA Virtual Configuration

4.3. AFTA Functional Overview

During normal operation, members of a virtual group communicate between themselves and with other virtual groups by passing messages through the virtual bus. The virtual bus can be modeled using an abstraction known as a Byzantine Resilient Virtual Circuit

(BRVC) [Har87]. This abstraction has several characteristics that make it suitable for use in a fault-tolerant system as described below.

- packet delivery is reliable, so a virtual group which sources a packet can expect delivery of the packet, assuming that the specified receiving virtual group exists.
- packets are delivered in the same relative order, i.e. if a virtual group sources two packets, packet A followed by packet B, destined for the same virtual group, the receiving virtual group receive packet A before it receives packet B.
- each member of a virtual group will receive packets in the same order as all other members of the virtual group.
- each functioning member of a virtual group will receive bitwise identical copies of every packet delivered to the group.
- packet delivery is synchronous among members of a virtual group.

The characteristics described above are used to implement various functions. These functions are designed to satisfy the requirements of Byzantine resilience.

Reliable message delivery is a requirement for a fault-tolerant system, such as the AFTA. Building reliable message delivery on top of an unreliable packet delivery system is tedious and can never guarantee complete coverage for all random faults [BG87]. By providing reliable packet delivery, there is no need for additional software protocols to ensure reliable message delivery.

Another requirement of fault-tolerant systems is to guarantee consensus among functioning members of a fault-masking group. This requirement is met by voting packets in the Network Element. The NE will perform a source congruency on single-source data. Relative packet ordering is also necessary to guarantee consensus.

Fault-tolerant computers must also be synchronized. Synchronous packet delivery is used as a method of synchronizing processors in a virtual group. A group can synchronize itself by sending a packet to itself, then waiting for that packet to be delivered. A timeout is used so that if a member of a group does not respond within an allowed period the same way as the majority of the group members, that member will be ignored and the remaining members will continue uninhibited. This type of synchronization is known as functional synchronization.

The Network Element implements several support operations in addition to the packet delivery functions.

One of the most important support functions is system reconfiguration. The AFTA supports dynamic reconfiguration, allowing the grouping of physical processing sites into virtual groups to be changed in real-time. The mechanism for reconfiguring the system is the CT update. The configuration table, or CT, is a table stored internally on the Network Element. The processors have no direct access to the CT, but they can effect changes in the CT using the CT update. Processors may keep copies of the CT in local processor memory. In previous FTTP designs, a single distinguished virtual group referred to as the reconfiguration authority was given sole authority for performing the CT update process. While a reconfiguration authority-based protocol may be used for certain AFTA reconfiguration modes, there is no hardware support planned in the AFTA for enforcing a single reconfiguration authority. However, the NE only permits fault-masking groups to perform a CT update.

The Network Element contains a global synchronous timer which is synchronized to the fault-tolerant clock (FTC). This timer is used as the basis for calculating timeouts by the scoreboard and for providing timestamps on packets. Because the timer is synchronized to the FTC, the value can be considered congruent among all FCRs. The timer is initialized to zero upon system reset and is realigned by voting during the reintegration process.

Another support function is initial synchronization, or ISYNC. When the AFTA is first powered up, the Network Elements are not synchronized. ISYNC is the procedure by which the Network Elements become synchronized. The two subsections of the NE that require synchronization are the fault-tolerant clock and the global controller. The fault-tolerant clocks, designed using standard FTC techniques, will become synchronized automatically within 190 μ s. The global controllers, however, must explicitly synchronize themselves. The ISYNC procedure involves continual exchanges, using the 2 round source congruency exchange, to determine which NEs are ready to synchronize.

Recovery of a Network Element following a transient fault in the NE uses a process similar to the ISYNC function. An NE that is trying to recover performs continual 2 round exchanges. The remaining NEs in the working group reintegrate the recovering NE by performing a single 2 round exchange. If the working group detects the recovering NE, the NEs are resynchronized and realignment is initiated. During the realignment process, the configuration table and global synchronous timer are exchanged and voted. This operation ensures that the state of the newly reintegrated NE is consistent with the rest of the system. At the conclusion of realignment, the recovering NE is considered completely reintegrated with the working group.

The Network Element is also responsible for monitoring certain aspects of the system to aid in the diagnosis of faults. The Network Element maintains bit vectors, called syndromes, to indicate when certain unusual behavior is observed. These syndromes can indicate problems with virtual group members, Network Elements, or FCR interconnects. The syndromes are delivered with each packet exchanged by the virtual bus. Some of the syndromes apply specifically to the associated packet; others are simply an accumulation since the last packet delivery. Note that all Network Elements will not necessarily see the same error conditions, therefore the syndromes must be treated as single-source data.

4.4. AFTA Network Element

The Network Element is the core of an AFTA cluster. The Network Element connects on one side to a number of processing sites, and on the other side to the other Network Elements in the cluster. The ensemble of Network Elements forms a virtual bus network through which the processors communicate.

4.4.1. Network Element Addressing Convention

An applications program for the AFTA can almost always ignore the physical AFTA configuration, and use only the virtual configuration as the programming model. Using the virtual configuration, there is no need to refer to a specific Network Element or FCR, since these concepts only exist in the physical configuration. Systems programs (including device-drivers) however, must occasionally refer to a specific Network Element or FCR. Examples of these situations include reading or writing I/O devices, performing system re-configuration (via CT updates), and diagnosing of faults.

The method of referencing an NE or FCR is known as Network Element addressing. Each Network Element is assigned a unique ID number. The NEIDs are assigned as shown in Figure 4-3. Note that successive NEIDs can be found in a counter-clockwise pattern around the ensemble of NEs.

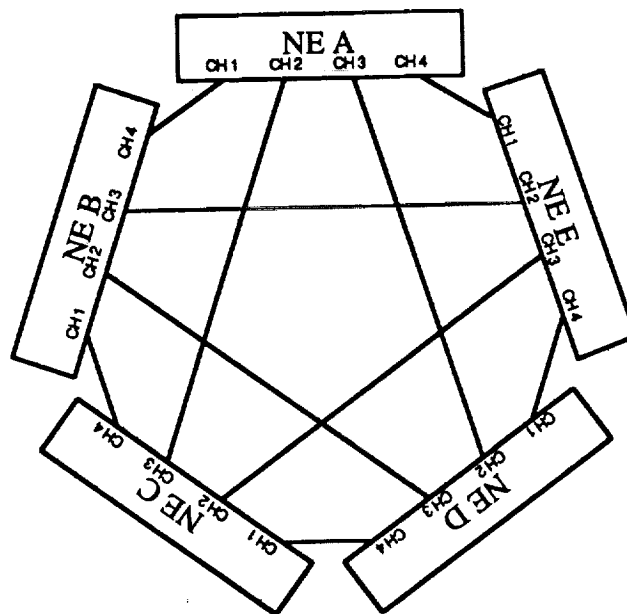


Figure 4-3. Network Element Addresses

FCR IDs are taken from the NEID of the Network Element in the FCR. Henceforth, the terms FCR ID and NEID will be synonymous.

Absolute addressing refers to a specific Network Element based on its position as observed from outside the cluster. For absolute referencing (i.e. NE A, NE D, etc.) the IDs are represented numerically as follows:

A = 0
B = 1
C = 2
D = 3
E = 4

Relative addressing is used to refer to a Network Element based on the referenced NE's position relative to the local NE. In previous versions of the FTPP, relative addressing used the form of left, right, opposite, mine. To provide consistent referencing for the AFTA, with up to 5 NEs, relative addressing is determined by assigning the address CH 0 to the local NE. Then, continuing counter-clockwise beginning with the NE to the right of the local NE, successive relative addresses are assigned to each NE. Relative IDs are represented numerically as follows:

CH 0 = 0
CH 1 = 1
CH 2 = 2
CH 3 = 3
CH 4 = 4

By convention, this document uses the letter designations (i.e. NE A, NE B, etc.) to indicate the absolute address, and the numerical channel designation (CH 0...CH 4) to indicate the relative address. However, the AFTA hardware uses 3-digit binary numbers as outlined above for both addresses.

For situations where bit patterns represent masks or error syndromes, the numerical address represents the bit position within the byte. For example, an absolute mask for the AFTA has the form:

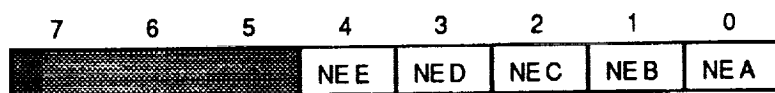


Figure 4-4. Absolute Mask

A relative mask for the AFTA has the form:

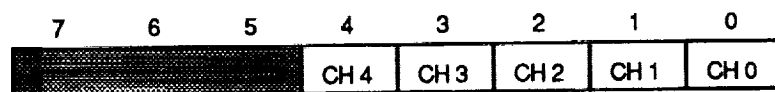


Figure 4-5. Relative Mask

To allow for ease of future expansion, unused bit positions are left undefined rather than used for packing multiple masks or error syndromes into a single byte.

The absolute numerical address can be derived from the relative address by the following:

$$\text{absNEID} = (\text{relNEID} + \text{myNEID}) \% \text{numNEs}$$

Conversion of bit patterns from relative to absolute reference can be accomplished by the following (assuming that undefined bit positions are cleared):

$$\text{absMASK} = (\text{relMASK} \ll \text{myNEID}) | (\text{relMASK} \gg (\text{numNEs} - \text{myNEID}))$$

4.4.2. Network Element Functional Description

This section describes the functions of the AFTA Network Element. The functions provided by the Network Element include the primary data exchange primitives and the secondary system maintenance primitives.

4.4.2.1. Data Exchange Primitives

The AFTA Network Element provides a number of data exchange primitives for the Processing Elements to use. The primary use of the primitives is to transfer data from one virtual processing site to another. The primitives are also be used to vote common-source data or distribute single-source data within a virtual processing site. The primitives also have various side effects, including synchronization, time stamping, and syndrome reporting. A special set of primitives are provided which produce side effects that directly affect the state of the Network Element. These special primitives include CT updates, transient NE recovery, and voted resets. Most of these primitives are solely for the use of the AFTA operating system. When the application program requests inter-VG communication, the AFTA operating system transparently maps the task's communication request to the appropriate data exchange primitive in the manner described in Section 5.

The processor uses the same procedure to access any of the primitives. Data is transferred from a physical processor to the associated Network Element through the processor's output buffers. First, the processor must select a contiguous segment of 64 bytes within the output data block. Next, the segment is filled with the 64 bytes of data (unless the class 0 primitive is being used). Then, the buffer descriptor located in the output info block is filled with the appropriate information. The output info block specifies the primitive to be executed, the destination virtual group, and the location of the data in the output data block. Finally, the ownership of the output buffer is transferred to the Network Element by performing the send operation on the ring buffer manager.

The Network Element transfers data to processors through the input buffers. The Network Element selects the next free cell in the ring buffer for the processor and fills the data and info fields in that cell. Then, the cell is enqueued for ownership by the processor. The processor must access the cells in the order in which ownership is transferred from the NE to the PE to preserve total packet ordering. When the PE detects a cell which it owns in its input buffers, the PE must transfer the data and descriptor information from the cell into local memory and return ownership of the cell to the Network Element. The emptying of input buffer cells must be a high priority operation since the longer buffers are allowed to remain full, the more likely flow control will be asserted.

The processors use the ring buffer manager to control ownership and determine ownership status of its buffer cells.

The processor uses a send operation to transfer ownership of an output buffer cell, and a return operation to transfer ownership of an input buffer cell. Note that the processor can only relinquish ownership of cells to the NE; the processor does not have the capability to overtly acquire ownership of cells.

The status of a processor's buffers is determined by either the ready operation, for output buffers, or the next operation, for input buffers. Each of these operations returns a pointer to the buffer cell that must be used in the appropriate ring buffer. In the current NE implementation, the output buffer only has one cell, so the ready operation always returns a pointer of 0. The next operation returns a pointer between 0 and 31, inclusive. Each operation also returns an invalid bit that, when clear, indicates that the pointer is valid. If the processor does not own any cells in the associated buffer, the invalid bit will be set.

4.4.2.1.1. Class 0

The class 0 primitive is used when only the side effects of a data primitive are needed. The class 0 does not exchange any data. When a virtual group executes a class 0 primitive, all of the descriptor information in both the output and input info blocks is defined. All of the information is valid, except for the vote syndrome, which is undefined.

The data in the output data block does not need to be defined, but the pointer in the output info block must point into the processor's own output data block. The data in the input data block is not guaranteed to be congruent among members of the destination virtual group and must be ignored.

4.4.2.1.2. Class 1

The class 1 primitive performs a single round of exchange and vote on data from a fault-masking virtual group (FMG). Only FMGs are allowed to execute the class 1 primitive, since at least 3 independent copies of data are required for an unambiguous bitwise majority vote.

The output data block of the source virtual group contains the copy of data to be voted from the local processor. The input data block of the destination virtual group contains the voted result. The contents of the input data block may be considered congruent among members of the destination virtual group.

4.4.2.1.3. *Class 2*

The class 2 primitive performs a two round, or source congruency, exchange on data from any virtual group. The source of the data may be a simplex virtual group or a single member of a fault-masking group. The class 2 primitive is the only mechanism by which simplexes are allowed to communicate with other virtual groups.

The output data block of the source virtual group member contains the data to be distributed by the class 2 primitive. The data in the output data blocks belonging to virtual group members who are not sourcing data is ignored by the Network Element. The input data block of the destination virtual group contains the exchanged data. The contents of the input data block may be considered congruent among members of the destination virtual group.

4.4.2.1.4. *Broadcasts*

Broadcasts are a useful means of transmitting data to all active virtual groups in the cluster. Broadcasts are more of a drain on system resources than the standard point-to-point communication primitives, so only FMGs are allowed to send broadcasts. The use of broadcasts should be minimized.

The broadcast primitive is invoked as a modifier to the existing exchange primitives. Any of the primitives, including the data exchange and the special primitives, can be delivered as a broadcast. If a broadcast is used, the ToVID field in the output info block is ignored. A virtual group can determine that a received packet was delivered as part of a broadcast by examining the broadcast modifier bit in the class field of the input info block. The contents of the input data block can be considered congruent among all operational PEs in the cluster, unless the packet was delivered as part of a class 0 broadcast primitive.

4.4.2.2. *Configuration Table Updates*

The Network Element must keep track of the grouping of physical processors into virtual groups. The NE uses a data structure known as the configuration table, or CT, to contain this mapping. The CT also contains information for timeouts and vote masks. The CT is modified whenever any of this information must be changed. The CT update primitive is used to update the CT in a synchronous and atomic manner. Only a fault-masking group is allowed to execute the CT update primitive, unless there are no FMGs in the cluster. The

CT update must be done with a class 1 exchange class, unless a simplex is performing the CT update under the previous exception.

The configuration table on the NE consists of a number of entries, with each entry corresponding to a potential virtual group. Not all of the potential virtual groups will necessarily be active at any one time. The CT update primitive modifies a subset of the CT entries in one atomic action. Up to eight CT entries can be modified with a single CT update primitive, enough to allow one complete quadruplex to be formed or disbanded from or to its constituent simplex virtual groups. Multiple virtual groups can be formed or disbanded by one or more successive CT updates.

Each entry in the CT update packet is a direct replacement for the selected entry in the configuration table. The entry contains the VID number, which selects the CT entry to be updated. The entry also specifies the redundancy level of the new virtual group, a mask detailing which members of the virtual group are considered to be functional (for data voting), and the value to be used for timeouts on the virtual group. Finally, the entry contains a list of the physical processors that make up the virtual group. The list specifies the Network Element ID to which the processor is connected and the buffer set the processor uses to communicate with its Network Element. The list is only as long as is indicated by the redundancy level of the entry.

4.4.2.3. Initial Synchronization

Initial synchronization, or ISYNC, is the procedure by which the Network Elements become synchronized at power-up or after a system reset. The ISYNC process can be initiated by either a processor connected to an NE or by the global controller on the NE. The latter is only an option for systems in which the microcode is non-volatile.

The following indicate the expected condition of the system after power-up.

- FTCs self-synchronize within S seconds
- $\leq F$ FCRs are faulty
- all functioning FCRs were powered-up or reset within T seconds of each other (i.e. maximum skew is T seconds.)
- Cardinality and connectivity requirements to survive F Byzantine faults are satisfied.

From the above assumptions, it is clear that the system meets the requirements for Byzantine resilience if a means for exchanging single source data is provided. However,

the skew is unacceptable for operational mode. The purpose of ISYNC is to reduce the skew to acceptable levels.

The condition of being in ISYNC mode is treated as a piece of single source data. By exchanging this data through a source congruency, each FCR reaches a consensus about the relative synchronization state of the cluster.

A message string indicates the synchronization state of the cluster. Each FCR has one entry in the message string. Nominally the message string is:

$m_0 m_1 m_2 m_3 m_4$

which indicates that all FCRs are ready to synchronize. The presence of message m_i ($m_i = 0x0AEC6BF0$, for all i) indicates that FCR i is ready to synchronize. A message x_i , where $x_i \neq m_i$, indicates that FCR i is not ready to synchronize. All non-faulty FCRs which are not ready to synchronize are designed to send x_i rather than m_i . An example of a message string which indicates that only FCRs 1 and 4 are ready to synchronize is:

$x_0 m_1 x_2 x_3 m_4$

Each message in the message string is a single-source message from the respective FCR. Consequently, the messages must be exchanged using a 2-round source congruency algorithm.

The algorithm for performing ISYNC is as follows. At startup the ISYNC message string is exchanged using a 2 round exchange. Each FCR broadcasts the message x_i until the ISYNC initiator requests message m_i . The ISYNC initiator must wait at least $(T+S)$ seconds after power up before requesting m_i . An ISYNC timeout with a timeout period $\geq T$ is started after $2F+1$ valid messages are observed in the message string. ISYNC is terminated when one of the following conditions is met:

- all valid messages are observed in the message string
- the ISYNC timeout period expires.

Figure 4-6 illustrates the timeline for 5 Network Elements performing the ISYNC algorithm.

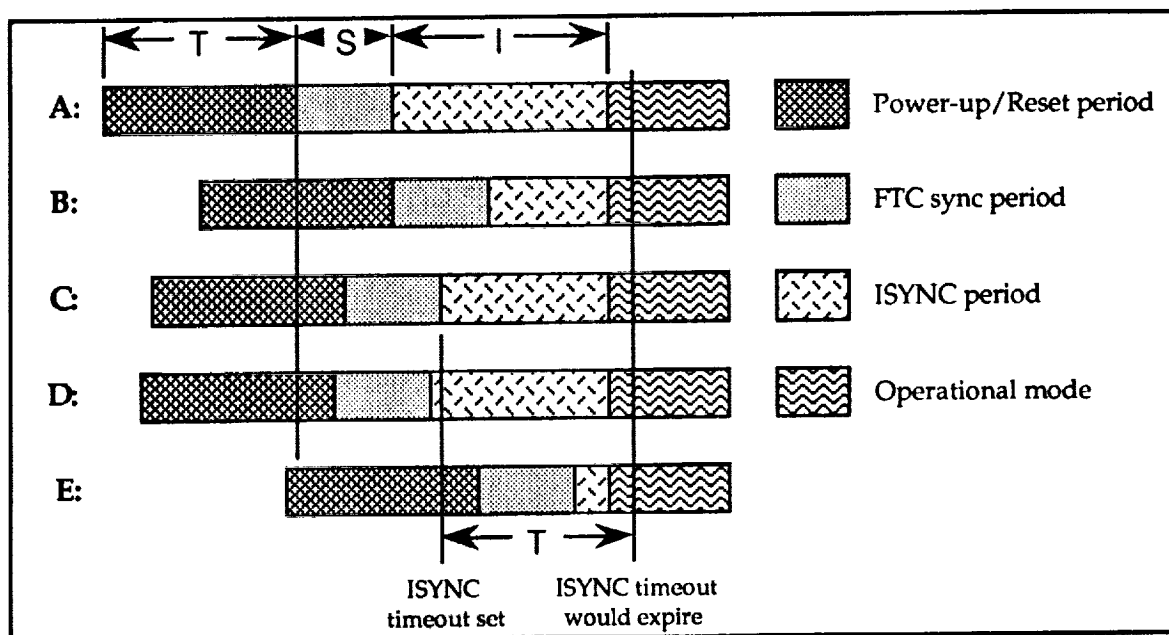


Figure 4-6. ISYNC Procedure

ISYNC is attempted for a period of $2T$ seconds. If ISYNC has not succeeded by this time, the NE terminates ISYNC and enters the transient NE recovery procedure. The transient NE recovery procedure is described in Section 4.4.2.4.

4.4.2.4. *Transient NE Recovery*

The transient NE recovery procedure, or TNR, is similar to the ISYNC procedure. Transient NE recovery is used to reintegrate a failed NE. An NE which has suffered a transient failure may not have any permanent faults that prevent it from functioning as a member of the cluster. However, the NE must be resynchronized and realigned with the working group before it can be declared non-faulty.

An NE which has suffered a transient failure may have been reset by a voted reset, the watchdog timer, or the power-on reset. The NE will enter ISYNC mode as a part of the boot procedure. This NE has no way of knowing that a working group exists. It will attempt to perform exchanges of the ISYNC message string as defined in Section 4.4.2.3. Since the working group is performing other packet exchanges, the ISYNC procedure will fail. After a period of $2T$, the failed NE will terminate ISYNC and enter TNR.

The TNR procedure is similar to the ISYNC procedure. The same message string is used to determine whether or not a particular NE is in TNR or not. The major difference is that TNR is terminated immediately whenever any new Network Elements are observed in

the message string. The current NE mask (as defined in Section 4.4.3.3.2) is used as a reference for the message string. For this reason, the working group must set their NE mask to reflect the current working group configuration before executing TNR.

The failed NE enters TNR after failing ISYNC. The working group enters TNR when a virtual group requests to send a transient NE recovery (TNR) packet. A single TNR packet exchange is performed, and the resulting message string is examined by the NEs to determine if any new NEs, as compared to the NE mask, are present. If no new NEs are observed, the functioning NEs return to the operational mode unscathed. Upon entering TNR from ISYNC, a Network Element remains in the TNR state indefinitely unless and until a successful TNR exchange is observed. If the TNR operation is successful, as indicated by one or more new NEs observed in the message string, the CT is exchanged and voted into the newly recovered NE(s).

Successful completion of TNR requires alignment of the state of the reintegrated NE to reflect the state of the working group. The configuration table is aligned by exchanging and voting the entire contents of the CT. Timeouts in the scoreboard are aligned by resetting all timeouts, which effectively restarts all packet ready and flow control conditions. The global synchronous timer is realigned by exchanging and voting the timer value. One consequence of voting the timer is that the timer value effectively stops incrementing until the realignment of the timer is complete. The packet buffers in the recovered NE are set to their initial condition after power up.

Normally all PEs in the reintegrated NE are declared as inactive spares (i.e. simplexes). The state of these PEs is assumed to be the same as a freshly powered-up processor. These spares can be integrated into an existing virtual group by a CT update following successful completion of TNR. Alignment of these PEs with an existing virtual group is the responsibility of the reconfiguration authority task. The reconfiguration authority is also responsible for broadcasting the current system configuration to these new PEs.

The functioning NEs assume that the recovering NE has resynchronized its FTC to the remaining ensemble and has entered TNR mode. For this reason, a functioning ensemble must wait at least 2T seconds after performing a voted reset before attempting TNR.

4.4.2.5. Voted Resets/Monitor Interlocks

The AFTA architecture is designed to tolerate any single random fault. In addition, the system is designed to allow attempts to restart a failed element or reconfigure around the

failed element. Voted resets are a mechanism to attempt recovery from transient faults, and monitor interlocks are one mechanism for reconfiguration around permanent faults. Voted resets and monitor interlocks are exchanged along with data exchanges on the interconnecting fiber optics. No additional interconnects are required to distribute this information.

If a virtual group detects a Processing Element or Network Element in error, the virtual group may try to restart the element by asserting a voted reset. A voted reset can be directed to either a single Processing Element within an FCR, or to the entire FCR. Since resetting a Network Element also resets all processors attached to the Network Element, there is no separate provision for just resetting a Network Element. Only a fault-masking group is allowed to assert a voted reset, unless there are no FMGs in the cluster. The voted reset primitive must be done with a class 1 exchange class, unless a simplex is performing the voted reset under the previous exception.

When an entire FCR is lost due to some sort of catastrophic failure, all of the I/O on the lost FCR becomes unavailable. Critical I/O is usually replicated in multiple FCRs, so that a one FCR can take over I/O activity for another FCR if the second FCR fails. The first FCR must obtain access to the I/O device, and the second FCR must be disabled from driving the device. A monitor interlock can be used to attempt to turn off the driver in the failed FCR to allow the new FCR to assume ownership of the device. Only a fault-masking group is allowed to assert a monitor interlock, unless there are no FMGs in the cluster. The voted reset primitive must be done with a class 1 exchange class, unless a simplex is performing the monitor interlock under the previous exception.

The built-in support on the Network Element for voted resets and monitor interlocks includes a special packet type for executing the primitives and a set of discrete outputs for invoking the desired side effect. The discrete outputs are only asserted for one FTC cycle following the exchange of the voted reset or monitor interlock packet. Additional support circuitry may be required to interface the discrete output to processing or I/O elements.

4.4.2.6. Syndrome Reports

The Network Element places syndromes in the input info block whenever a packet is successfully delivered to a virtual group. The syndromes must not be considered congruent across all members of a destination virtual group.

The syndromes indicate various anomalies the Network Element observes during the execution of an exchange primitive. The syndromes can be divided into two major classes: NE syndromes and scoreboard syndromes.

The NE syndromes indicate any anomalous behavior detected anywhere on the NE except within the scoreboard. The NE syndromes, located in the second longword of the input info block buffer cell, include indications of vote errors, fault-tolerant clock synchronization errors, and fiber-optic link errors.

The vote syndrome indicates that one or more channels did not agree with the final voted result. For a class 1 exchange, a vote syndrome means that the channel did not produce the expected output. For a class 2 exchange, a vote syndrome means that the channel did not forward the correct value during the second exchange round. Since the second round of a class 2 is completely contained within the Network Element, the Network Element is indicted by a vote syndrome on a class 2 exchange. Either the processor or the Network Element is indicted by a vote syndrome on a class 1 exchange. The vote syndrome is undefined for a class 0 exchange.

The clock and link syndromes are not necessarily associated with the packet on which they are delivered. Each syndrome represents an occurrence of the indicated error at some time between delivery of the previous packet and delivery of the current packet.

The clock syndrome indicates that the rising edge of the fault-tolerant clock (FTC) signal from the associated channel did not fall within the acceptable skew with reference to the local FTC signal. Since the local channel is always synchronized with itself, the clock syndrome bit corresponding to the local channel, bit 0, will always be zero.

The link syndrome indicates that a violation was reported by the TAXI receiver chip for the associated channel. A violation indicates that the TAXI received an invalid pattern over the fiber-optic interface. Violations are usually a sign of catastrophic failure in the affected Network Element or a break in the physical fiber-optic link. Since there is no TAXI receiver chip associated with the local channel, the link syndrome bit for the local channel, bit 0, will always be zero.

The scoreboard syndromes indicate anomalous behavior detected by the scoreboard during SERP processing. The scoreboard syndromes, located in the third longword of the input info block buffer cell, include indications of scoreboard vote errors, OBNE timeouts, and IBNF timeouts.

The scoreboard vote syndrome indicates that one or more channels of the source virtual group did not agree with the voted result for the class, destination VID, or user byte. Data in the SERP is exchanged using a 2 round exchange, voted in the data paths, and voted again in the scoreboard. The scoreboard vote syndromes only indicate vote errors detected during voting in the scoreboard; there is no indication of vote errors during data path voting of the SERP.

The OBNE timeout syndrome indicates that the associated virtual group member(s) did not place a packet in their output buffers within the timeout skew of the majority of the virtual group members. The timeout skew is specified by the timeout field in the CT entry. All members are expected to transmit a packet simultaneously, within the timeout skew. If a majority, but not a unanimity, of virtual group members are observed with packets in their output buffers, a timeout is initiated. If the timeout expires before the other members transmit the packet, the remaining members are ignored, the packet is exchanged, and an OBNE timeout is recorded.

The IBNF timeout syndrome indicates that the associated virtual group member(s) did not deassert flow control on their input buffers within the timeout skew of the majority of the virtual group members. The timeout skew is specified by the timeout field in the CT entry. All members are expected to free space in their input buffers simultaneously, within the timeout skew. If a majority, but not a unanimity, of virtual group members are observed with deasserted flow control on their input buffers, a timeout is initiated. If the timeout expires before the other members empty their input buffers, the remaining members are ignored, flow control is deasserted for the virtual group, and an IBNF timeout is recorded.

4.4.2.7. Timestamps

The Network Element places a timestamp in the input info block for each packet successfully delivered to a virtual group. The timestamps are congruent across all members of the destination virtual group, and across all active processors in the case of a broadcast.

The timestamp is a 32-bit quantity that indicates the relative time within the cluster. An external time source (such as a GPS reference or a time-of-day clock built into the PEs) can be used to add a constant to the timestamp to gauge absolute time. The resolution of the timestamp value is 1.28 μ s. The maximum timestamp value is 4,294,967,295, or 0xFFFFFFFF, which corresponds to approximately 5500 seconds. When the timestamp

counter reaches the maximum value, it wraps around to 0 with no indication to the processor. The processors must obtain timestamps often enough to detect wraparound.

The timestamp counter is initialized to zero during ISYNC and increases monotonically thereafter, except during transient NE recovery (TNR). When a Network Element is reintegrated using TNR, the timestamp counters are realigned as part of the recovery process to ensure congruency of the timestamps. This realignment causes the timestamp counters to cease increasing until the realignment is complete. Consequently, the timestamps after TNR will be slightly smaller than they would be if TNR were not performed. The processors can correct this error by estimating the timestamp error by measuring the duration of TNR using internal timers and applying a correction constant to the new timestamps.

4.4.2.8. NE Debug Commands

The following describes the debug commands supported by the Network Element debugger. These commands are implemented in a special version of the microcode. The debug commands can be used to debug new Network Element hardware and for performing stand-alone self-testing of the Network Element.

wrap_vme(byte1,byte2)-Copies *byte1* to the VDAT bus register, then copies the VDAT bus register to *byte2*.
wrap_serp(proc,serp_entry)-Generates the SERP entry for *proc* and returns it in *serp_entry*.
wrap_to_input(pack,proc)-Copies *pack* (64 byte packet) to the selected input buffer.
reflect_from(channel)-Reflects the packet stored in the selected channel to the VDAT bus. Any channels which have their debug routers enabled or are connected to a fiber-optic wrap path will receive the reflected data.
vote_deliver1(proc)-The packet in the data path FIFOs is voted and stored in the input buffer selected by *proc*. Voting rules for Class I (voted) exchanges are used (PE mask ANDed with NE mask)
vote_deliver2(proc)-The packet in the data path FIFOs is voted and stored in the input buffer selected by *proc*. Voting rules for Class II (source congruency) exchanges are used (NE mask with source masked out).
clear_output(proc)-Clears the selected processor's output buffer.
clear_input(proc)-Clears the selected processor's input buffer.
clear_datap(channel)-Clears the selected channel's data path FIFO.
write_pattern(pattern)-Writes a pre-determined byte pattern into *pattern*.
write_pe_mask(mask)-Writes *mask* to the PE mask register in the data path voter.
write_ne_mask(mask)-Writes *mask* to the NE mask register in the data path voter.
ct_enter(ctentry)- The configuration table entry specified by *ctentry* is copied into the CT on the NE. A scoreboard CT update is not performed.
ct_update(ctentry)-The configuration table entry specified by *ctentry* is copied into the CT on the NE. Then, a scoreboard CT update is performed.
mask_update(mask)-Updates the NE mask in the configuration table.

process_serp(serp,class,proc,fromvid,tovid,pemask,nemask)-*serp* is entered into the DPRAM of the scoreboard. Then, the global controller requests the scoreboard to process *serp*. The results of the processing are returned to the debugger through the VME DPRAM.

next_message(class,proc,fromvid,tovid,pemask,nemask)-The scoreboard is requested to continue processing a previously entered *serp* and return the next message ready to be sent.

lerp_to_dpram(lerp)-Generates a LERP message for the local NE and copies it to the data DPRAM. This LERP is never exchanged or used by the scoreboard.

return_time(timestamp)-Returns the 32-bit value stored in the global synchronous timer.

write_debug_enables(enablepattern)-Selects each debug router (for external channels only) to receive data either from the external interface (via the TAXI receiver for that channel) or from the local data source (the VDAT bus).

infinite_loop()-The global controller jumps to a single-state infinite loop. The watchdog timer is not kicked during the loop, so the NE should be reset by the watchdog timer after the timeout period has expired.

4.4.3. Network Element Programming Reference

This section describes the procedures for accessing the Network Element from the Processing Elements. The section defines the memory map of the NE, the format of data and control registers on the Network Element, and packet formats used by the data exchange primitives to transmit data between virtual groups via the Network Element.

4.4.3.1. Processor/Network Element Interface

Processors communicate with the NE via the VMEbus. The processors must be capable of performing VME bus master functions A24 and/or A32, and D32. The processors must also function as A16 and D08(o) slaves. Packet data is transferred by the processor to and from the Network Element using the processor master modes. The NE delivers mailbox interrupts to the processor using the slave modes.

The Network Element responds to both supervisor and user address modifier codes, allowing applications software to write directly to the Network Element. Should it be necessary to prevent user accesses, simple modifications can be made to the Network Elements so that only supervisor address modifiers are acknowledged.

Byte ordering in the NE VMEbus ports follows the Motorola convention, a.k.a. big endian, with the most significant byte in the lowest address. Any Processing Element selected for use in the AFTA is expected to comply with the byte ordering convention specified by the VMEbus.

The base address of the NE must be on a 64K byte boundary (i.e. lower 16 bits cleared). All accesses to the Network Element by the PEs are 32 bit cycles. Some NE ports may not define all 32 bits of the accessed word. In fact, some locations are accessed for their side-effects and have no data associated with them.

The Network Element has the capability of delivering a signal to the processors. The NE can perform D08(o) master cycles in the A16 address space of the VMEbus. The actual location and the data to be written can be specified for each processor. The location may be a memory location in the processor's RAM or it may be a register for a processor mailbox interrupt, depending on the Processing Element chosen. The signal can be delivered on packet transmission, packet reception, or on the input buffer full (IBF) condition. Microcode changes select the signal delivery condition.

4.4.3.2. Memory Map

A memory map of the NE as viewed from the VMEbus is shown in Figure 4-7. The memory map is divided into two main segments. The first is the data segment, which is used to transfer data between the processors and the Network Element. The data segment also contains status and control registers for each processor. The data segment maps into the DPRAM memory on the NE. The second segment is the buffer manager. The buffer manager regulates the use of the dual-port RAM to prevent contention for data resources between the processors and the NE.

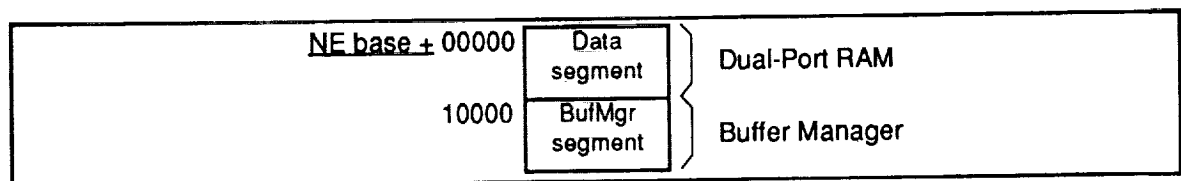


Figure 4-7. NE Memory Map

Each Network Element is attached to a maximum of 8 processors. Each processor has its own window of addresses in the DPRAM and its own set of ports in the buffer manager.

4.4.3.2.1. Data Segment

The data segment is implemented with 4 4K x 8 dual-port RAM devices. These devices provide data buffering between the NE and the PEs. Because it is dual-ported, each side can access the data segment asynchronously, provided that they do not access the same lo-

cation. Arbitration to prevent simultaneous access is performed by the buffer manager. A PE's adherence to the buffer manager arbitration must be ensured by the operating system message passing software, i.e. there is no hardware enforcement. System software on the PE must be written to follow the ownership rules of buffer cells in the data segment. Failure to adhere to the ownership rules described in Section 4.4.3.2.2 could result in overwriting an incoming packet or, in the worst case, the desynchronization of the NE.

The data segment is divided into equal sized windows for each of the maximum of eight processors per NE. Each processor has an identical structure superimposed on this window. The structure of the data segment is shown in Figure 4-8.

Each processor window in the data segment is divided into 5 blocks; four are used for packet transfer and one is unused. The four used blocks are paired into input and output data blocks. One of the blocks in each pair is used for actual data communication; these blocks are referred to as data blocks. The other blocks, known as info blocks, contain information pertaining to the data in the corresponding data block.

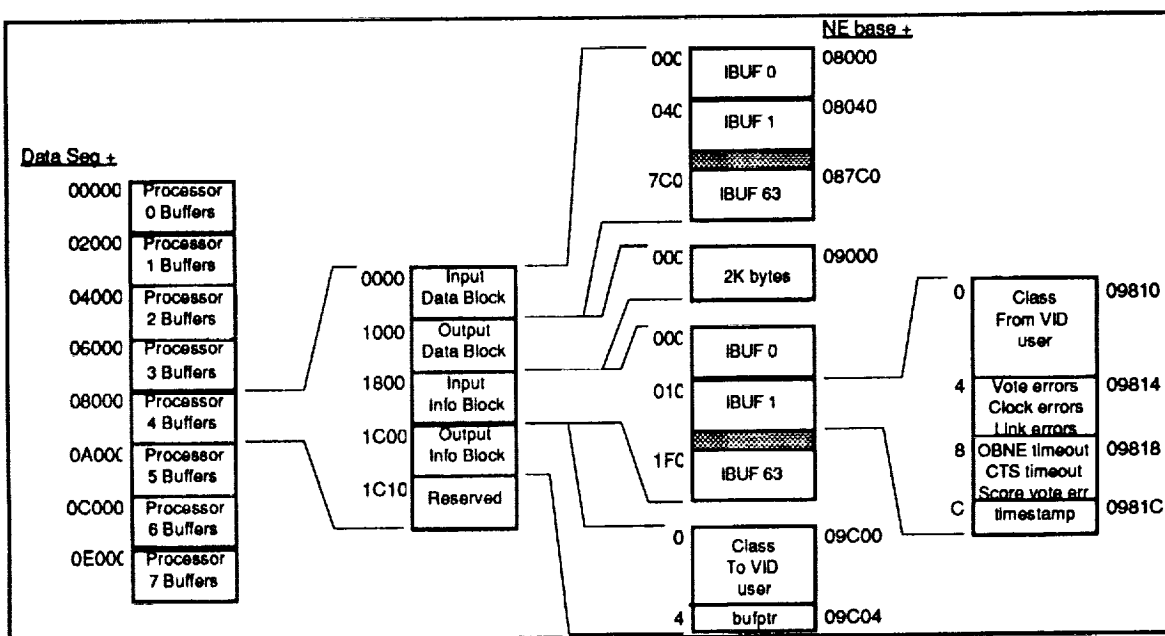


Figure 4-8. DPRAM Memory Map

4.4.3.2.1.1. Outgoing (Transmit) Buffering

The output ring buffer is used to send packets to the NE virtual bus. The output data block and the output info block comprise a ring buffer of 1 cell. The output ring buffer cell consists of 64 bytes in the output data block and 8 bytes in the output info block. Only the

locations in the output info block are associated a-priori with the buffer. The bufptr field in the info cell points to the beginning of the data cell in the output data block. The output data block is a flat memory-mapped section that can be used at the processor's discretion. The output data cell is considered to be a contiguous 64 byte block starting at the location indicated by the value of bufptr. If the output info cell is owned by the NE, the data cell is also considered to be owned by the NE. All other locations in the output data block are owned by the PE.

When the processor wants to send a packet to the NE, it first makes sure that the output buffer is empty by either polling the buffer manager waiting for the OBF (Output Buffer Full) bit to be deasserted (low), or by waiting for the packet transmit signal from the previous packet transmission (if the NE-PE signal is enabled for the packet transmit condition). Next, the processor finds an unused 64 byte cell in the output data block. A 64 byte block must be allocated even if no data is to be exchanged. The pointer to the cell is entered in the bufptr location of the output info cell. The header information is copied into the other locations in the info cell, and the data (if any) is copied into the data cell allocated above. Finally, the packet is sent by informing the buffer manager that the ring buffer cell contains a valid packet. The processor is informed, either through the buffer manager OBF bit or the packet transmit signal, when the packet is sent and the ring buffer cell can be used for another packet.

When the NE observes an output buffer which contains a valid packet, the NE ensemble determines, using the scoreboard, whether this packet represents a packet to be exchanged, voted, and delivered by the NE data path hardware. If the packet is validated by the scoreboard, the NE reads the packet from the buffer and returns ownership of the buffer to the processor. The processor must not access the buffer until it is returned by the NE.

4.4.3.2.1.2. Incoming (Receive) Buffering

The input ring buffer is used to receive packets from the NE virtual bus. The input data block and input info block together comprise a ring buffer of 64 cells. Each cell in the input ring buffer consists of 64 bytes in the input data block and 16 bytes in the input info block. Each cell in the input data block corresponds to a cell in the input info block.

Once a packet is validated for transmission, the NE exchanges and votes the packet. The voted packet is then delivered to the receiving virtual group's input buffer along with

the header information. When the NE wants to deliver a packet to a processor, it first obtains a ring buffer cell from the buffer manager. Then, the packet data is written into the proper data cell, and the header information is written into the info cell. Next, the NE transfers ownership of the buffer to the PE using the buffer manager. The buffer cell is considered owned by the processor until the processor explicitly returns ownership of the cell to the NE. Finally, if the packet receive signal is enabled, the NE sends a signal to the processors which make up the receiving virtual group.

When a processor observes a packet delivery, either by polling the IBE bit or by reception of the packet receive signal, the processor obtains the location of the new packet by reading the ready location in the buffer manager. The Ready IB field indicates the cell number of the oldest unread packet in the processor's input ring buffer. The processor copies the data and corresponding header information from the input ring buffer cell into the processor's local memory. When the processor is done with the ring buffer cell, the cell is freed for use by another packet by accessing the return location in the buffer manager.

Broadcast packets are delivered to all input ring buffers within an AFTA cluster, whether or not the associated processor is a member of an active virtual group.

4.4.3.2.1.3. Information Block Fields

The information blocks contain control and status information associated with data located in the data blocks. In previous versions of the FTTP, this information was either prefixed or appended to the packets in the data FIFOs, or entered into the class FIFO. A brief discussion of each field is contained below. In the case of the error fields, a set bit corresponds to an observed error, and a cleared bit corresponds either to observed normal behavior or to an undefined channel or FCR.

4.4.3.2.1.3.1. Class

The packet class selects the data exchange primitive to be executed by the NE. The packet class is a full 8 bit field, yielding a maximum of 256 different packet classes. Individual bit fields in the class field define particular aspects of the packet class as shown below in Figure 4-9.

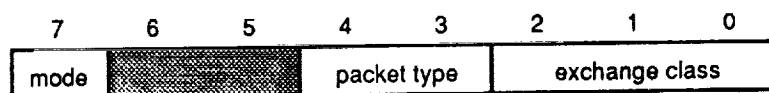


Figure 4-9. Packet Class Field

The exchange class defines the protocol to be used when exchanging the packet. Currently, the following values are valid:

- 0-class 0 (no data)
- 1-class 1 (one round exchange)
- 2-class 2 (two round exchange) from member on Network Element A
- 3-class 2 (two round exchange) from member on Network Element B
- 4-class 2 (two round exchange) from member on Network Element C
- 5-class 2 (two round exchange) from member on Network Element D
- 6-class 2 (two round exchange) from member on Network Element E

The packet type defines the contents of the ring buffer data cell. Data packets are the normal mode of communication between virtual groups. Data packets are treated as a contiguous stream of 64 bytes. There is no structure enforced by the NE on data packets. The other packet types, however, have specific formats that must be adhered to as described in Section 4.4.3.3. The following are the current valid packet types:

- 0-data
- 1-configuration table update
- 2-transient NE recovery
- 3-voted reset

The mode determines how the packet is to be distributed. Two modes are supported: normal (bit 7 is cleared) and broadcast (bit 7 is set). In the normal mode, the packet is delivered to the virtual group specified in the ToVID field. In broadcast mode, all processors (including the sender), regardless of whether or not they are a member of an active virtual group, will receive a copy of the packet. The ToVID field is ignored for broadcast packets.

Not all packet classes are allowed in all circumstances. The following outlines the packet exchange rules.

- Only a fault-masking group is allowed to send a packet with exchange class of 1.
- Only a fault-masking group is allowed to send a CT update packet.*
- The CT update packet must be exchanged using a class 1 (one round) exchange.*
- Only a fault-masking group is allowed to send a voted reset packet.*
- The voted reset packet must be exchanged using a class 1 exchange.*
- Any virtual group can send a class 0 or class 2 packet.
- Any virtual group can send a data packet or an isync packet.
- Any virtual group can send a normal packet.
- Only a fault-masking group is allowed to send a broadcast packet.*

* Unless the HLF (higher-life-form) bit in the scoreboard is not set.

Undefined fields and values in the packet class are reserved. Undefined fields must be set to zero.

4.4.3.2.1.3.2. *ToVID*

The ToVID is an 8-bit field specifying the virtual group to which the packet is to be sent. The VID numbers may range from 0 to 255. Not all VID numbers will be valid, since there will be at most 40 active virtual groups in the system. If the NE detects an attempt to send to a non-existent virtual group, the packet is removed from the sending virtual group's output buffer and discarded.

The ToVID field is ignored for broadcast packets.

4.4.3.2.1.3.3. *FromVID*

The FromVID field is an 8-bit field specifying the virtual group that sent the packet. The VID numbers may range from 0 to 255. This field is always valid.

4.4.3.2.1.3.4. *User Field*

The user field is an 8-bit field for arbitrary use by the processor. The value in the user field is exchanged and voted along with the SERP data. SERP voting rules are used on the user field instead of standard class 1 voting rules. The user field can be used to send out-of-band data between virtual groups.

4.4.3.2.1.3.5. *Vote errors*

Vote errors indicate if the data emanating from a participant during the packet exchange disagreed with the majority in any way. For class 1 packets (one round exchanges), the syndrome bits are only defined for NEs on which the virtual group has members. For class 2 packets (two round exchanges), the syndrome bits are defined for all NEs except the NE on which the source member resides. Undefined syndrome bits will be cleared. The format of the vote error field is shown in Figure 4-10. The vote error field is in the relative NE format.

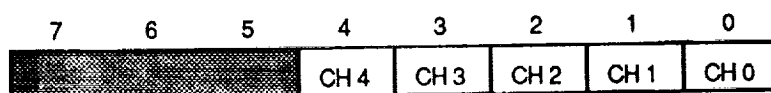


Figure 4-10. Vote Error Field

4.4.3.2.1.3.6. Clock Errors

Clock errors indicate that sometime since the last packet was exchanged by the NE, the FTC signal from the indicated NE fell outside the allowable skew window. A clock error signals a potential problem with the indicated NE or the cable linking the indicated NE with the local NE. The format of the clock error field is shown in Figure 4-11. The clock error field is in the relative NE format. The bit corresponding to the local NE (bit 0) is undefined.

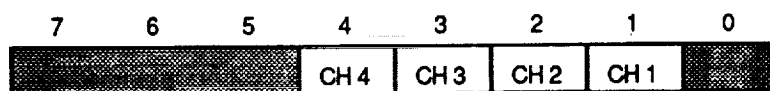


Figure 4-11. Clock Error Field

4.4.3.2.1.3.7. Link Errors

Link errors indicate that sometime since the last packet was exchanged by the NE, an error was detected on the indicated fiber-optic link. An error detected by the TAXI receiver devices is indicated by assertion of the VLTN (violation) signal, which usually indicates loss of synchronization with the transmitter. A link error signals a potential problem with the indicated NE or the cable linking the indicated NE with the local NE. The format of the link error field is shown in Figure 4-12. The link error field is in the relative NE format. The bit corresponding to the local NE (bit 0) is undefined.

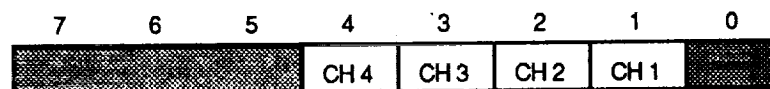


Figure 4-12. Link Error Field

4.4.3.2.1.3.8. OBNE timeout

The OBNE timeout (Output Buffer Not Empty) field indicates that the members of the source virtual group corresponding to the set bits did not request to send the packet within the allowable timeout skew. These members are considered desynchronized from the other members of their virtual group until a reintegration procedure is performed on the virtual group.

The format of the OBNE timeout field is shown in Figure 4-13. The OBNE timeout field is in the absolute NE format. Only bits corresponding to NEs on which the source virtual group has members are defined.

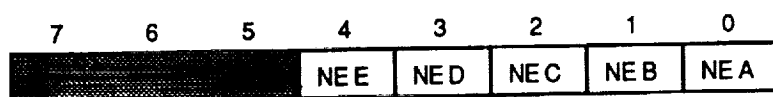


Figure 4-13. OBNE Timeout Field

4.4.3.2.1.3.9. IBNF timeout

The IBNF timeout (Input Buffer Not Full) field indicates that the members of the destination virtual group corresponding to the set bits did not free enough space in their input buffers to hold the incoming packet within the allowable timeout skew. These members are considered desynchronized from the other members of their virtual group until a reintegration procedure is performed on the virtual group.

The format of the IBNF timeout field is shown in Figure 4-14. The IBNF timeout field is in the absolute NE format. Only bits corresponding to NEs on which the destination virtual group has members are defined. The IBNF timeout field is undefined for broadcast packets.

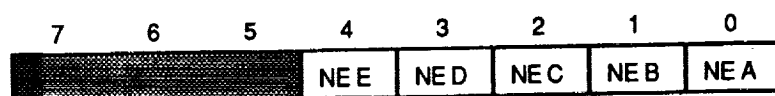


Figure 4-14. IBNF Timeout Field

4.4.3.2.1.3.10. Scoreboard Vote Error

A scoreboard vote error indicates that the corresponding virtual group member did not agree with the majority regarding the type of packet to be exchanged. Scoreboard vote errors are only collected on data contained in the SERP, which includes the packet class, the destination virtual group (ToVID field), and the user field.

The format of the scoreboard vote error field is shown in Figure 4-15. The scoreboard vote error field is in the absolute NE format. Only bits corresponding to NEs on which the source virtual group has members are defined. The scoreboard vote error field is generated by the scoreboard and reflects discrepancies observed by the the scoreboard during SERP processing. Vote errors occurring in the data path voters during the voting of the SERP during the second round of the SERP exchange are not detected.

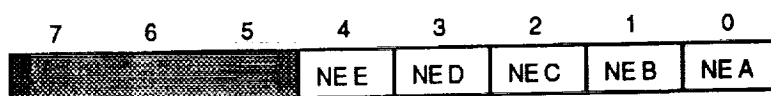


Figure 4-15. Scoreboard Vote Error Field

4.4.3.2.1.3.11. Timestamp

The timestamp field is a 32 bit field representing the time that the packet was exchanged. To be exact, it is the time at which the scoreboard determined that a valid packet condition existed to allow the packet to be exchanged. The timestamp value is determined from the global synchronous timer. This timer is initialized during ISYNC or recovery and increments synchronously with the FTC. The timer wraps around from 0xFFFFFFFF to 0x0 with no indication. The wraparound period is over 90 minutes, which should be plenty of time to detect wraparound in the operating system. The resolution of the least-significant bit of the timestamp is 1.28 μ s.

4.4.3.2.2. Buffer Manager

The buffer manager controls the status of the input and output buffers for each of the 8 processors connected to the NE. Processors use the output buffers to send packets through the Network Elements. The Network Elements use the input buffers to deliver packets to a processor.

Each buffer is owned by either the processor or the Network Element. Ownership depends on the type and the state of the buffer. A processor or the NE must only access buffers which it owns. Output buffers are initially owned by their associated processor, and input buffers are initially owned by the Network Element. A port can temporarily relinquish ownership of a buffer by invoking the SEND operation in the buffer manager. The buffer is returned to the original owner when the second port invokes the RETURN operation.

The buffer manager is accessed by the processor using the VMEbus. The processor status/control ports on the buffer manager are mapped to locations in the VMEbus address space as shown in Figure 4-16.

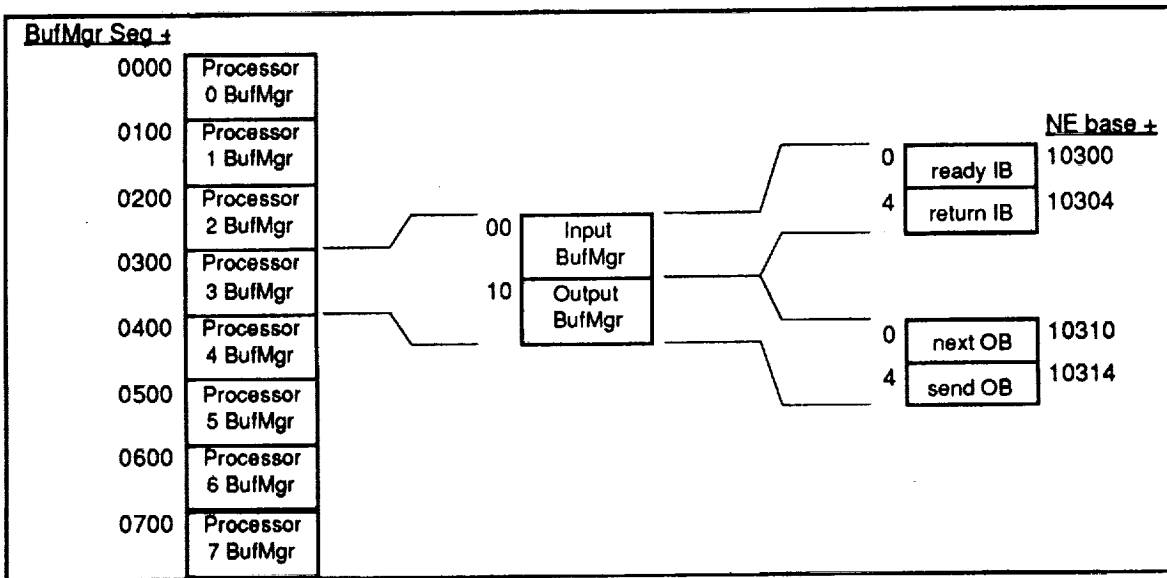


Figure 4-16. Buffer Manager Memory Map

For output buffers, the processor has access to the next and send ports. Reading the next port yields the OBF bit for the output buffer. Accessing (either reading or writing) the send port activates the SEND operation, which transfers ownership of the output buffer to the Network Element. The format of the next port is shown in Figure 4-17. The OBF (Output Buffer Full) bit is cleared if the buffer is empty and set if the buffer is full. The data read from or written to the send port is meaningless and must be ignored.

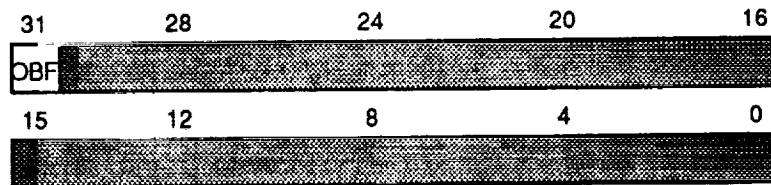


Figure 4-17. Next Port Format

For input buffers, the processor has access to the ready and return ports. Reading the ready port yields the IBE bit and the Ready-IB pointer. Accessing (either reading or writing) the return port activates the RETURN operation, which transfers ownership of the input buffer back to the Network Element. The format of the ready port is shown in Figure 4-18. The IBE (Input Buffer Empty) bit is cleared if the input ring buffer contains at least one unread packet and is set if the ring buffer is empty. The Ready-IB pointer indicates which ring buffer cell contains the oldest unread packet. The data read from or written to the return port is meaningless and must be ignored.

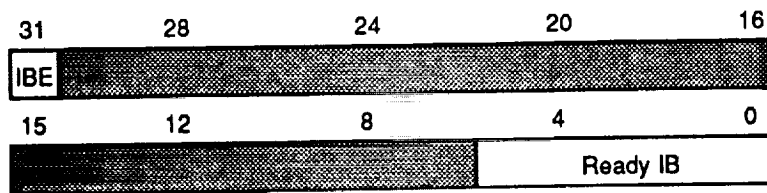


Figure 4-18. Ready Port Format

System software on the PE must be written to follow the ownership rules of buffer cells in the data segment. Failure to adhere to the ownership rules described above could result in overwriting an incoming packet or, in the worst case, the desynchronization of the NE.

4.4.3.3. Packet Formats

The following section defines the four types of packets that can be sent through the Network Element.

4.4.3.3.1. Data Packet

A data packet can be exchanged using any combination of the available exchange classes and modes. The format of a class 1 or a class 2 data packet is simply a contiguous string of 64 bytes. Any structure imposed on a data packet is done so by the Network Element driver software. A class 0 data packet has no data.

4.4.3.3.2. CT Update Packet

The CT update packet is used to modify the configuration table on the Network Elements. The configuration table (CT) is contained within the Network Element. The CT describes the mapping of physical processing sites into virtual groups. Processors make changes to the CT using the CT update packet.

The format of a CT update packet is shown in Figure 4-19. A CT update packet can update from 0 to 8 CT entries. Unused bits in the CT update entries (shown as shaded regions in the figures below) must be set to zero.

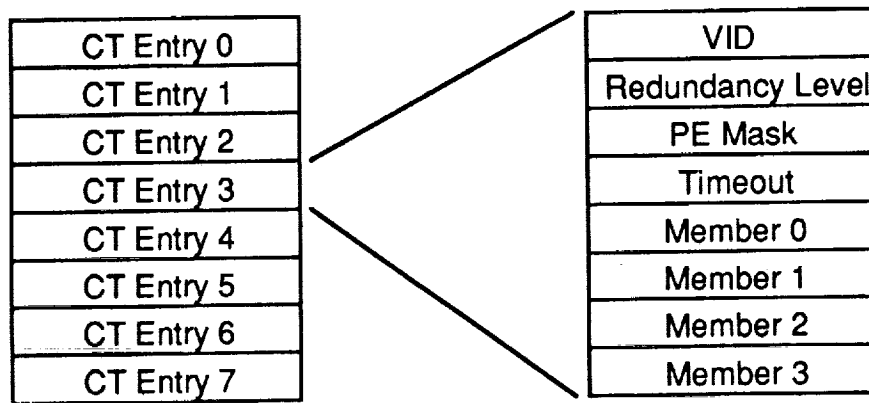


Figure 4-19. CT Update Packet Format

Each CT entry consists of 8 contiguous bytes. The first byte of the CT entry in the packet indicates which virtual group is to be updated. The next byte, shown in Figure 4-20, specifies the redundancy level to be used for the virtual group. The redundancy level (0-4) indicates how many members are to be included in the virtual group. A redundancy level of 0 indicates an inactive group.



Figure 4-20. Redundancy Level Field

The next byte, shown in Figure 4-21 is the PE mask. This field is used to mask out selected members of the virtual group during data voting. A set bit in the PE mask indicates that the corresponding member should be included in the data vote. Bits corresponding to NEs with no members in the virtual group must be cleared. The virtual group member is only masked during voting of data; the member is still considered during timeouts for the OBNE and IBNF conditions, and during scoreboard data voting. To eliminate the timeout penalty, the virtual group should be reconfigured to eliminate the faulty member either by reducing the redundancy level of the virtual group or by incorporating a spare processor to replace the faulty one.

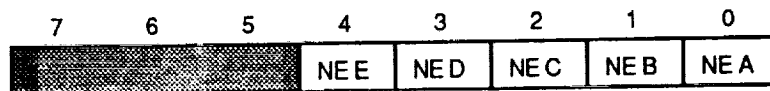


Figure 4-21. PE Mask Field

The next byte in the CT update entry is the timeout value. This value selects the timeout to be used for the virtual group when calculating the OBNE and IBNF conditions. The timeout is specified with a resolution of 1.28 μ s. The maximum timeout value is 326.4 μ s

(timeout field = 255). A timeout value of zero enables an infinite timeout for the virtual group. The timeout field is calculated using:

$$\text{timeout field} = \left\lceil \frac{\text{timeout value}}{1.28 \mu\text{s}} \right\rceil$$

Following the timeout byte is a list of the processors which make up the virtual group. The processors are specified in a format that uniquely identifies a single processor in the cluster. The format for the processor specification field is shown in Figure 4-22. The absolute NEID refers to the FCR containing the specified processor. The PEID refers to a single processor within the FCR. The processor listing starts with the 5th byte in the entry and continues until enough processors are specified to satisfy the redundancy level. Any extra bytes are unused. However, to avoid vote errors, these unused bytes must be defined.



Figure 4-22. Processor Specification Field

VID #255 is unique in that it refers to the Network Element rather than an ensemble of processors. Updating VID #255 in the CT packet is used to modify the NE mask. The NE reads a new NE mask from the location normally reserved for the PE mask. All other entries for VID #255 are unused. The format of the NE mask field is shown in Figure 4-23.

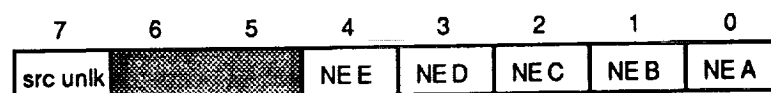


Figure 4-23. NE Mask Format

The bit field <4:0> in the NE mask performs the same function as the corresponding bit field in the PE mask. Setting a bit in this bit field enables the Network Element. Clearing the bit disables the Network Element's data and clock inputs. If a Network Element is disabled, all processors connected to that Network Element are also disabled, even if their PE mask bit is set.

The src unlk bit in the NE mask allows the source of a two round (or source congruency) exchange to be enabled on voting of the packet. Enabling the source during voting of source congruency packets when the system is in a degraded mode (3 or fewer Network Elements) allows additional anticipated failures to be tolerated. In a fault-masking mode (either 4 or 5 Network Elements enabled), the src unlk bit must always be cleared. Setting this bit violates the rules for Byzantine resilience and may allow single point failures to dis-

rupt the system. However, in a degraded mode, Byzantine resilience is undefined, so setting the src unlk bit is permissible.

A CT update packet must have exactly eight valid CT entries. CT entries can be repeated in the CT update packet with no undesirable (or noticeable) consequences. Also, unused virtual groups (those with a redundancy level of 0) can be used to pad the CT update packet to a full 8 entries.

4.4.3.3.3. *Transient NE Recovery Packet*

The TNR packet is used to invoke the transient NE recovery (TNR) procedure. The TNR procedure is used to reintegrate a desynchronized FCR. NEs which are reset (by a watchdog timer, voted reset, or other means), suffer a power supply interruption, or are powered on after the other NEs have completed ISYNC enter the TNR phase during the boot procedure. By performing transient NE recovery, a working group of NEs can synchronize a new NE with the working group.

The TNR procedure is invoked by sending a TNR packet. The format of a TNR packet for transmit is undefined. A TNR packet cannot be used to send user data to another virtual group. A TNR packet can be of any exchange class, and can be either normal or broadcast mode. Regardless of the exchange class or the data specified in the output data buffer, 64 bytes, as described in Figure 4-24, will be delivered to the recipient virtual group(s).

The first 5 entries (4 bytes each) of the TNR receive packet are the TNR messages as sourced by each Network Element. The expected TNR message for each NE is 0x0AEC6BF0.

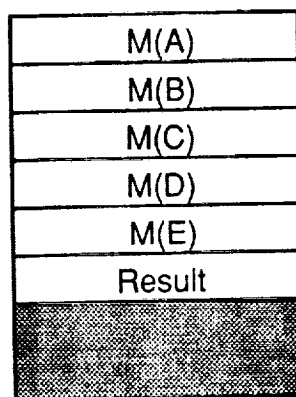


Figure 4-24. TNR Receive Packet Format

The next entry in the TNR receive packet is a byte which indicates which NEs sourced the expected TNR message. If the bit in the result byte corresponding to a particular NE is set, the message entry for that NE should be the expected TNR message.

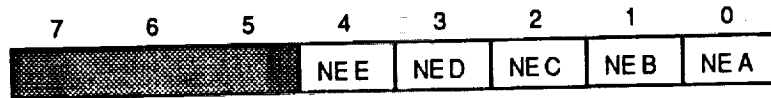


Figure 4-25. TNR Result Byte Format

The bytes in the TNR receive packet following the result byte are undefined.

4.4.3.3.4. Voted Reset Packet

The VRESET packet is used to perform the voted reset or monitor interlock operation. Voted resets enable a working group of FCRs to reset selected pieces of hardware in another FCR, presumably one which has suffered a transient fault or otherwise lost synchronization with the working group. The voted reset operation assumes that certain parts of the FCR to be reset are functional. Therefore, the voted reset function is a best-effort function; it is not guaranteed to work in all situations. There is no risk of catastrophic failure to the FCRs performing the voted reset.

The format of the VRESET packet is shown in Figure 4-26. The transmit packet and receive packet are identical. The first byte of the packet contains the VRESET command. The remaining 63 bytes in the packet are user defined.

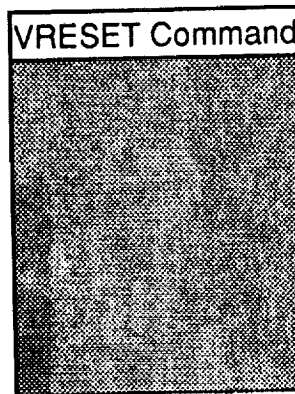


Figure 4-26. VRESET Packet

The VRESET packet is always exchanged using a class 1 exchange protocol. Specifying a class 0 or class 2 exchange class in the packet class field is illegal. Only fault-masking groups are allowed to source a VRESET packet. The packet is delivered to the destination virtual group (or all virtual groups, in the case of a broadcast) as any other standard class 1 data packet. The VRESET command is also loaded into the VRESET transmitter after being

voted on all NEs. The contents of the VRESET transmitter are sent to the NE to be reset on the next falling edge of the FTC. The VRESET command is deleted from the transmitter on the rising edge of the FTC.

The format of the VRESET command byte is shown in Figure 4-27. The FCR bit field indicates whether a single element (bit=0) or the entire FCR (bit=1) is to be reset. The NEID field selects one of the 5 FCRs, using the absolute addressing mode, on which to perform the voted reset. Valid values for the NEID field range from 0 to 4. The PE/IOC bit field selects whether a PE (bit=0) or an IOC (bit=1) is to be reset or interlocked, respectively. The element number field selects one of the processors or I/O elements to be reset. The element number field and the PE/IOC bit must be set to zero if the FCR bit is set. Undefined VRESET commands are ignored.

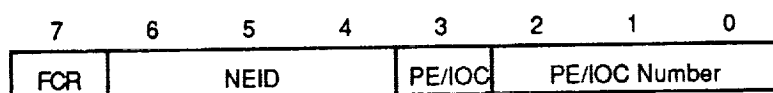


Figure 4-27. VRESET Command Byte

4.5. AFTA Component Physical Descriptions

The AFTA is designed for application in hostile embedded environments, such as military avionics bays, ground vehicles, or launch vehicles. As such, the AFTA is designed to meet stringent military requirements for such environments. The military specifies a number of standards that can be used to build military-qualified hardware. Two examples are the Standard Army Vetrionics Architecture (SAVA) for use in ground vehicles and the Joint Integrated Avionics Working Group (JIAWG) advanced avionics architecture (A3) for aircraft. This report studies AFTA designs based on these two standards. It must be noted, however, that the selection of these standards is made for concreteness of presentation; the AFTA design is not irrevocably tied to any such decision, nor is any endorsement by CSDL of these standards for use in the AFTA to be implied.

The AFTA is composed of either four or five fault containment regions (FCRs), each of which is housed in a line replaceable unit (LRU). The terms LRU and FCR are used interchangeably in this report. Figure 4-28 depicts a block diagram of an FCR. The LRU enclosure comprises a Faraday cage with environmental and EMI-resistant gaskets on all access hatches and ports. Each LRU contains a number of line replaceable modules (LRMs). An LRM is either a Processing Element (PE), network element (NE), input/output controller (IOC), or power conditioner (PC). Except for the power conditioner, an LRM is usually comprised of a single circuit board.

The LRMs are interconnected by a backplane bus for data exchange and power distribution. The PEs access the AFTA Network Element and I/O controllers over the bus. The backplane bus may use redundancy (for instance, a dual PI-Bus) for additional FCR reliability, or a split bus (for example, a VMEbus with VME Subsystem Bus (VSB)) for enhanced throughput. For a SAVA-based AFTA, the backplane bus is the System Backplane Bus (SBBUS) based on the VMEbus specification; a JIAWG-based AFTA uses the PI-Bus.

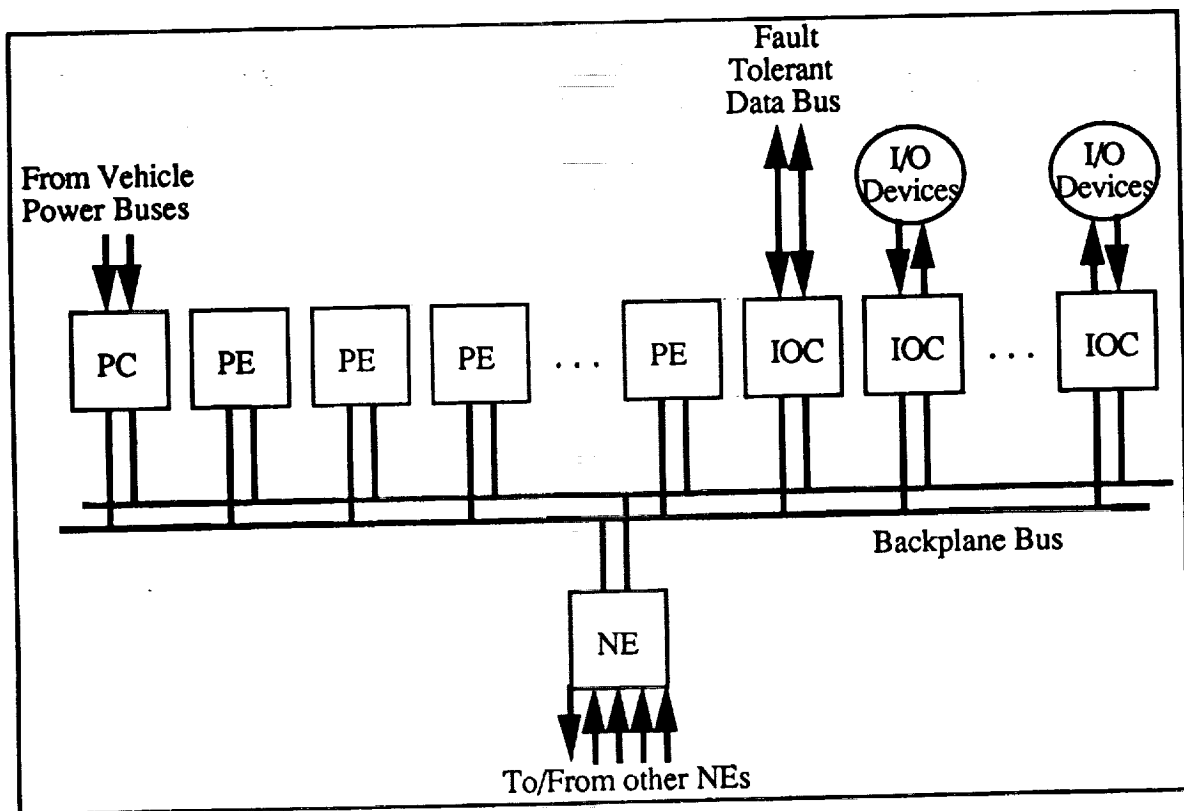


Figure 4-28. AFTA FCR Architecture

Hardware faults are isolated to an LRM using the fault detection, identification, and testing methods detailed in Section 5.6. Each LRM is separately removable from the AFTA according to the maintenance procedure outlined in Section 7. Live insertion or removal of an LRM depends on whether or not the backplane bus in use in the particular AFTA implementation supports live insertion/removal. If live insertion/removal is not possible, the FCR of which the LRM is a part must be powered down before replacing the LRM. The LRMs and LRUs are packaged for exposure to a forward operating environment to permit field replacement.

Figure 4-29 shows the overall dimensions of a SAVA-based AFTA FCR (LRU). This diagram does not show the power conditioners. Note the fiber optical bundle emanating from the connectors on the front of the enclosure.

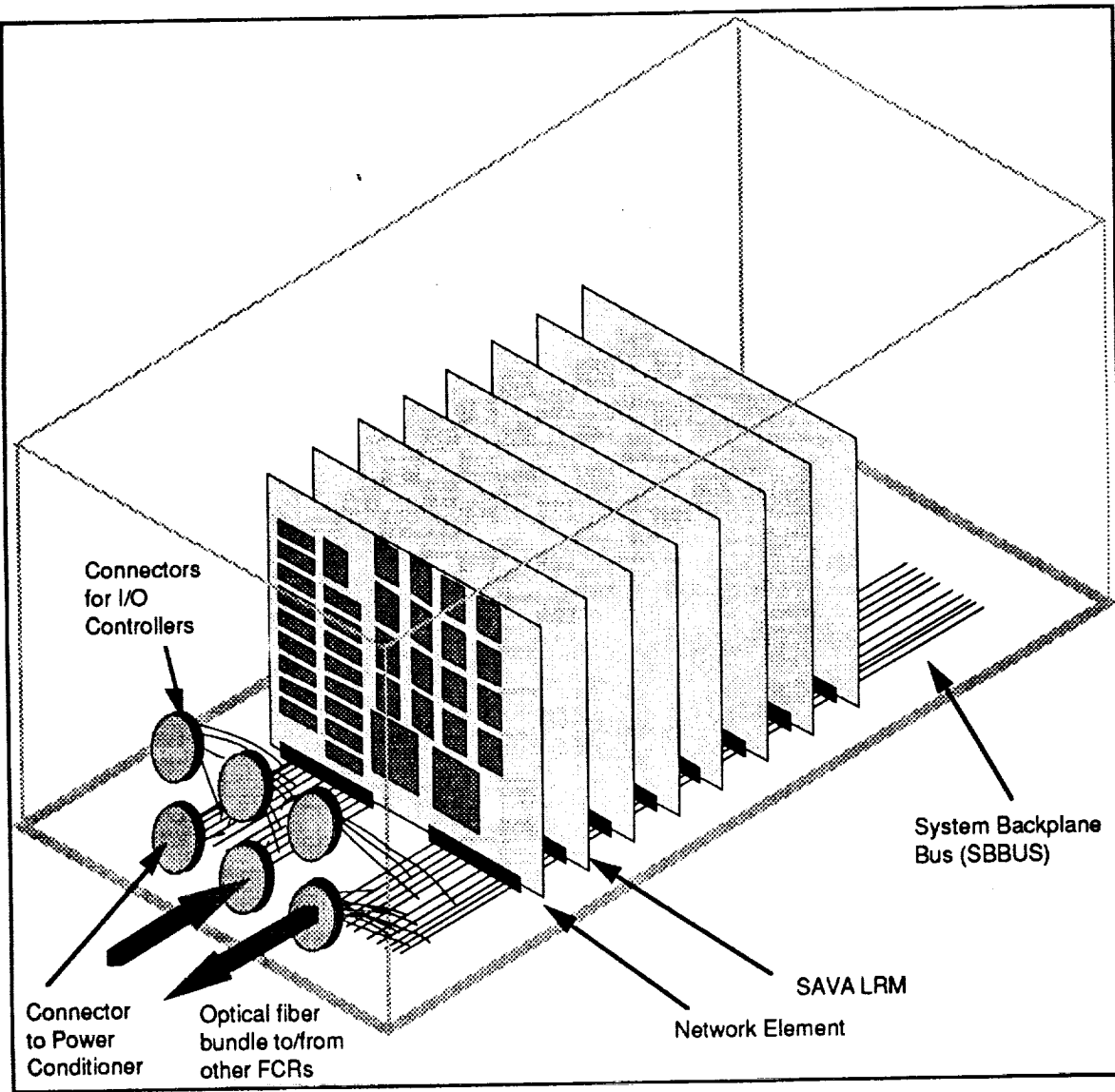


Figure 4-29. SAVA-based AFTA FCR

Figure 4-30 shows the overall dimensions of a SAVA-based single-card LRM. The 96-pin DIN connectors (3 rows) may be modified to contain 4 rows (128 pins) if the LRM is to reside in an SBBUS section of the FCR; the polyimide multi-layer board (MLB) may be changed as well depending on military qualification criteria.

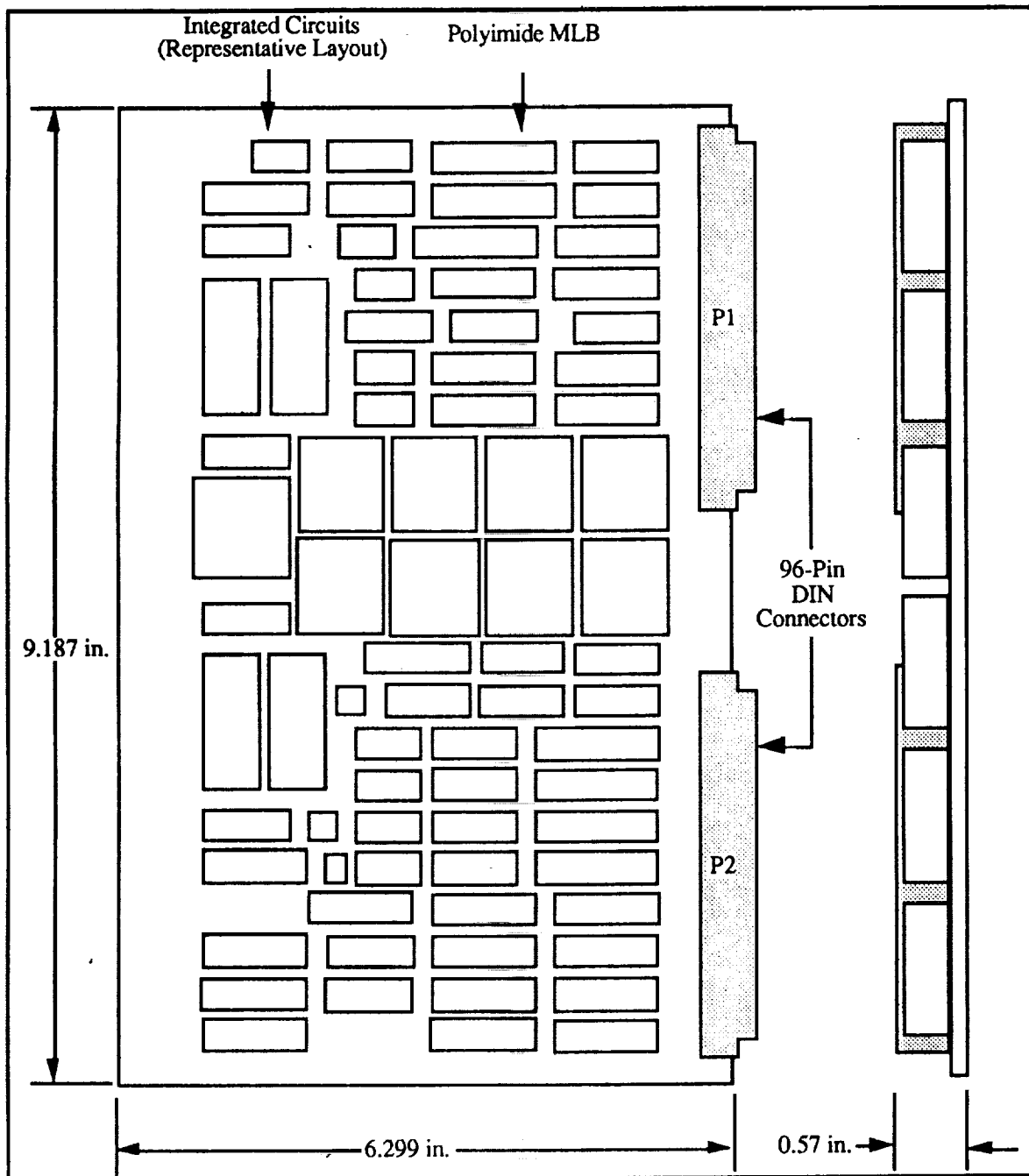


Figure 4-30. SAVA-based AFTA LRM

For comparison purposes, Figures 4-31 through 4-32 depict the physical dimensions of an AFTA fabricated to the JIAWG A3 (version 3.1) standards. The main differences visible at the current level of detail, besides differences in the bus interface, are that the LRMs are double-sided SEM-E cards with 250 pins available for connecting to the FCR back-

plane. Figure 4-31 shows an LRU based on this packaging standard, while Figure 4-32 shows the dimensions of the LRM.

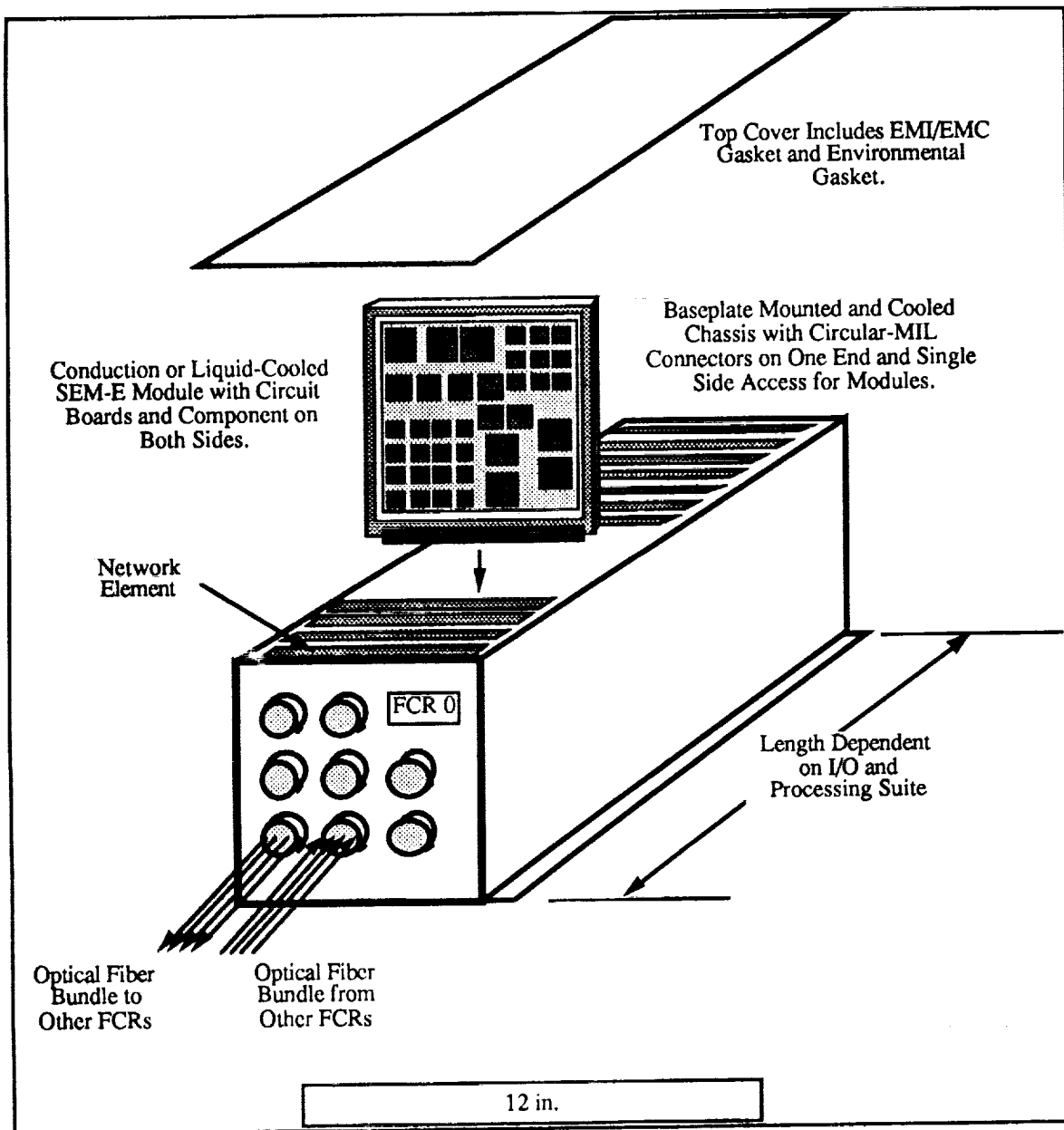


Figure 4-31. JIAWG-based AFTA FCR

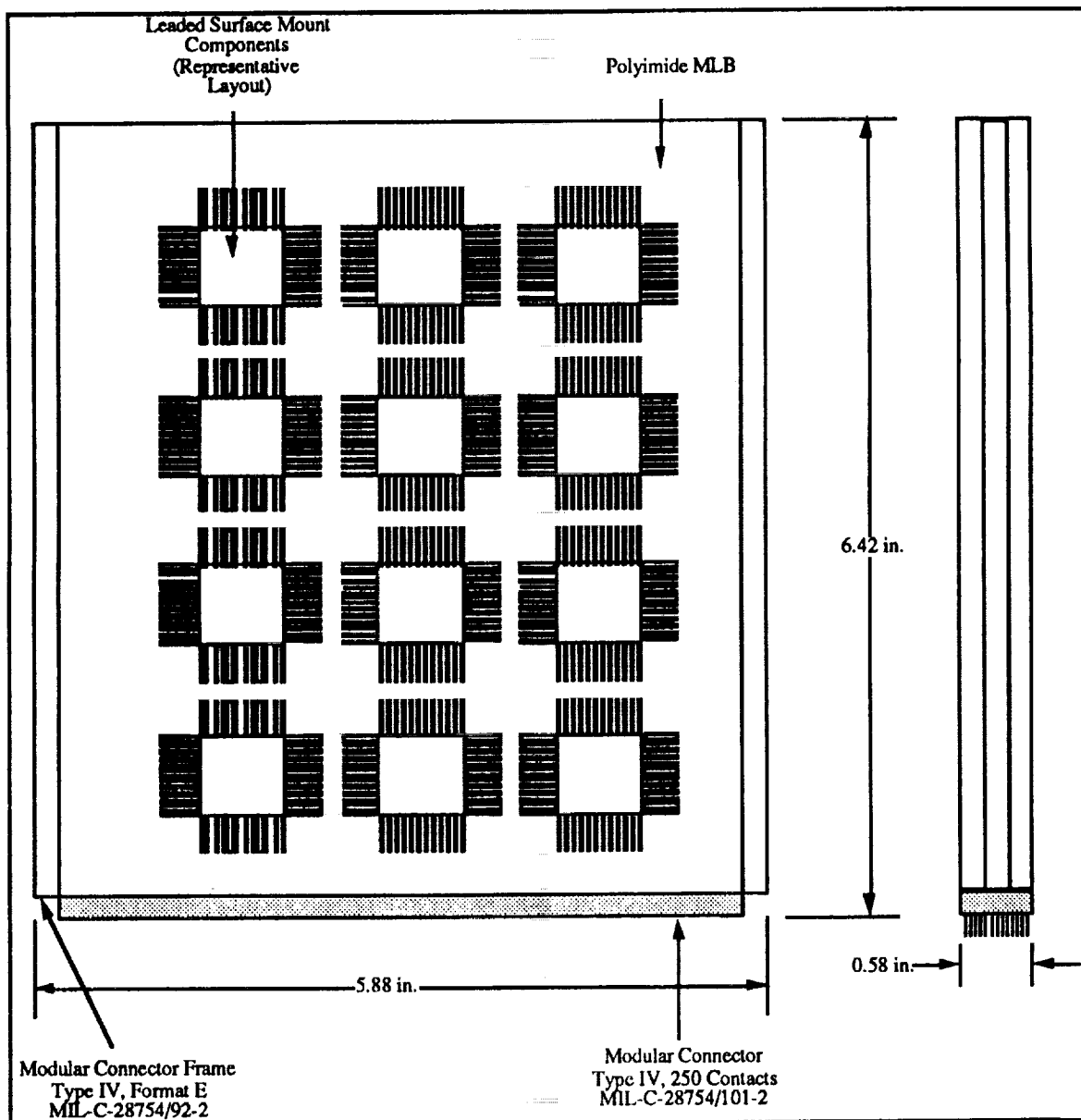


Figure 4-32. JIAWG-based AFTA LRM

4.5.1. Processing Element (PE) Characteristics

The Processing Elements (PEs) are the computational sites in the AFTA. Multiple PEs can be grouped to form a virtual simplex processing site for increased reliability. Non-critical tasks can be executed by a single Processing Element to maximize utilization of the processing resources. The mapping of PEs into virtual groups, or VGs, is maintained by the Network Elements. The mapping can be changed in real-time upon a request from the PEs.

The AFTA may contain more than one type of PE in any given AFTA implementation. The PEs for the AFTA are generally assumed to be available as non-developmental items (NDI), and may vary widely from implementation to implementation. The choice of PE and instruction set architecture (ISA) are completely up to the user of the AFTA. A functional block diagram of the typical components comprising a generic AFTA PE is shown in Figure 4-33. The PE contains a central processing unit (CPU) with an optional floating-point unit; multiple CPUs may reside on the PE LRM and may or may not comprise members of an AFTA VG, at the user's option. Typically, a local bus is used to communicate with on-board RAM, ROM, and I/O devices. Timers and oscillators are provided to generate a local time-of-day clock, time interval measurements, and timer-based interrupts. Optional local I/O may reside on the PE LRM. A system bus interface is used to gain access to the NE over the backplane bus.

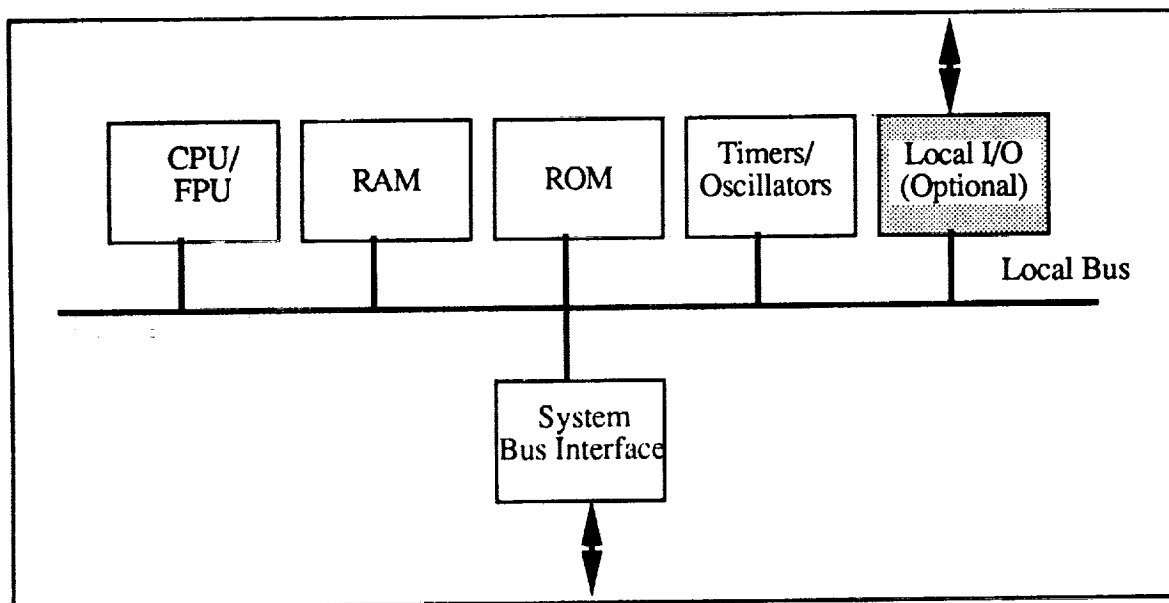


Figure 4-33. Functional Block Diagram of an AFTA Processing Element

For illustration purposes, three NDI PEs have been selected for use in the AFTA:

Radstone PMV 68M CPU-3A
 Lockheed Sanders STAR MVP
 SAVA GPPM

It should be stressed that selection of these PEs is made for concreteness of presentation; the AFTA is not irrevocably tied to any such selection, nor is any endorsement by CSDL of these PEs for use in the AFTA to be implied. It should be borne in mind at all times that the AFTA architecture is very flexible; a wide range of PEs can be used in the

AFTA, both in initial installations and in P³Is. Selection of appropriate Processing Elements for a particular AFTA application should be made based on throughput, commonality, compatibility, and other requirements as dictated by the particular situation.

Relevant characteristics of the selected PEs were gained from preliminary engineering documentation provided from the vendors and do not reflect commitments on the part of CSDL or the vendors.

The Radstone PMV 68M CPU-3A single board computer [Rad90] has the characteristics listed below.

Processor Type	25 MHz 68030/68882 FPU				
Throughput	<u>DAIS</u> 2.576M*	<u>Whetstones</u> 5.13M*	<u>Dhrystones</u> 8.95M*	<u>VUPS</u>	<u>MIPS</u> 8†
Memory	1.5 Mbyte SRAM, 512KByte EPROM				
Weight (Estimated)	2 pounds				
Power (Estimated)	25W				
Volume	Height	9.187 inches			
	Depth	6.299 inches			
	Thickness	0.063 inches (board)			
		0.800 inches (front panel)			
Failure rate	16,982h MTBF at Ground, Mobile, 45°C				
Operating Temperature Range	-55 to +85°C				
Storage Temperature Range	-62 to +125°C				
Relative humidity (Operating)	0% to 95%, MIL-STD-810D Method 507.4 Procedure III				
Cooling Requirements	Conduction cooling through thermal management layer to short card edges; wedge-lock connection to ATR enclosure				
Cost (1991)	\$23K (in quantity)				

Table 4-1. Characteristics of Radstone PMV 68M CPU-3A Processing Element

* Calculated by Draper.

† Obtained from vendor literature.

The Lockheed Sanders STAR MVP single board computer [San90] has the characteristics listed below.

Processor Type	25 MHz R3000/R3010 FPU				
Throughput	<u>DAIS</u>	<u>Whetstones</u>	<u>Dhrystones</u>	<u>VUPS</u> 20†	<u>MIPS</u>
Memory (typical)	16 Mbyte DRAM, 256KByte Cache, 1MByte EPROM				
Weight	2 pounds				
Power	20W				
Volume	Height	9.187 inches			
	Depth	6.299 inches			
	Thickness	0.063 inches (board)			
		0.800 inches (front panel)			
Failure rate	32,000h MTBF at Airborne, Uninhabited, 40°C				
Operating Temperature Range	-54 to +55°C, continuous				
Storage Temperature Range	-62 to +85°C				
Relative humidity (Operating)	100%, condensing				
Cooling Requirements	Conduction cooling through thermal management layer to short card edges; wedge-lock connection to ATR enclosure				
Cost (1991)	\$29K (quantity 1)				

Table 4-2. Characteristics of Lockheed Sanders STAR MVP Processing Element

† Obtained from vendor literature.

The SAVA General Purpose Processing Module (GPPM) [MIL-STD-344] has the characteristics listed below.

Processor Type	16 MHz 68020/68881 FPU				
Throughput	<u>DAIS</u> 1.03M*	<u>Whetstones</u> 2.05M†	<u>Dhrystones</u> 3.58M†	<u>VUPS</u>	<u>MIPS</u> 3.2††
Memory	128KByte SRAM, 128KByte EEPROM, 64KByte ROM, 4KByte DPRAM				
Weight	<2.25 pounds				
Power	<15W				
Volume	Height	9.187 inches			
	Depth	6.299 inches			
	Thickness	0.063 inches (board)			
		0.800 inches (front panel)			
Failure rate	>31,000h MTBF at Ground, Mobile, 85°C @ module edge				
Operating Temperature Range	-31 to +78°C (NB: evidently inconsistent with failure rate spec)				
Storage Temperature Range	-57 to +85°C				
Relative humidity (Operating)	94% @ 149°F				
Cooling Requirements	Conduction cooling to short card edges				
Cost (1991)	\$10K				

Table 4-3. Characteristics of SAVA GPPM Processing Element

* Calculated by Draper.

† Measured using XDAda compiler.

†† Obtained from vendor literature.

4.5.2. Network Element (NE) Characteristics

The Network Elements (NEs) form the core of the AFTA. Each FCR must contain an NE; the NE connects the FCR to all other FCRs in the AFTA cluster. The NEs include hardware to implement functions necessary for the Byzantine resilient properties of the AFTA. These functions include data exchanges, synchronization, syndrome recording, and monitor interlocks.

The Network Element design presented in this section is designed for use with the VMEbus backplane bus. Thus, the NE is compatible with both commercial and military grade VMEbus systems. The NE is also compatible with the SAVA SBBUS, with the simple replacement of the 96-pin Eurocard connectors specified by the VMEbus with the 128-pin SAVA backplane connectors. It must be noted, however, that the selection of the VMEbus is made for concreteness of presentation; the AFTA design is not irrevocably tied to this selection. Since the brassboard version of the NE is being built with a VMEbus interface, compatibility with military VMEbus systems and SAVA systems is virtually free. The NE can be used with other bus interfaces (PI-Bus, for example) with the expenditure of additional design effort.

4.5.2.1. Network Element Overview

A functional block diagram of the NE is depicted in Figure 4-34. The NE is divided into six major subsections: the VMEbus interface, the NE data paths, the inter-FCR communication system (IFC), the fault-tolerant clock (FTC), the global controller (GC), and the scoreboard. Most of these sections are tied together by the VDAT bus. This bus is used to transfer data from the dual-port buffer RAM to the IFC transmitter, from the voter output to the dual-port buffer RAM, and from the voter output to the scoreboard. The VDAT bus is also used by the global controller to load initializing parameters into the scoreboard, fault-tolerant clock, vote mask, and ring buffer manager.

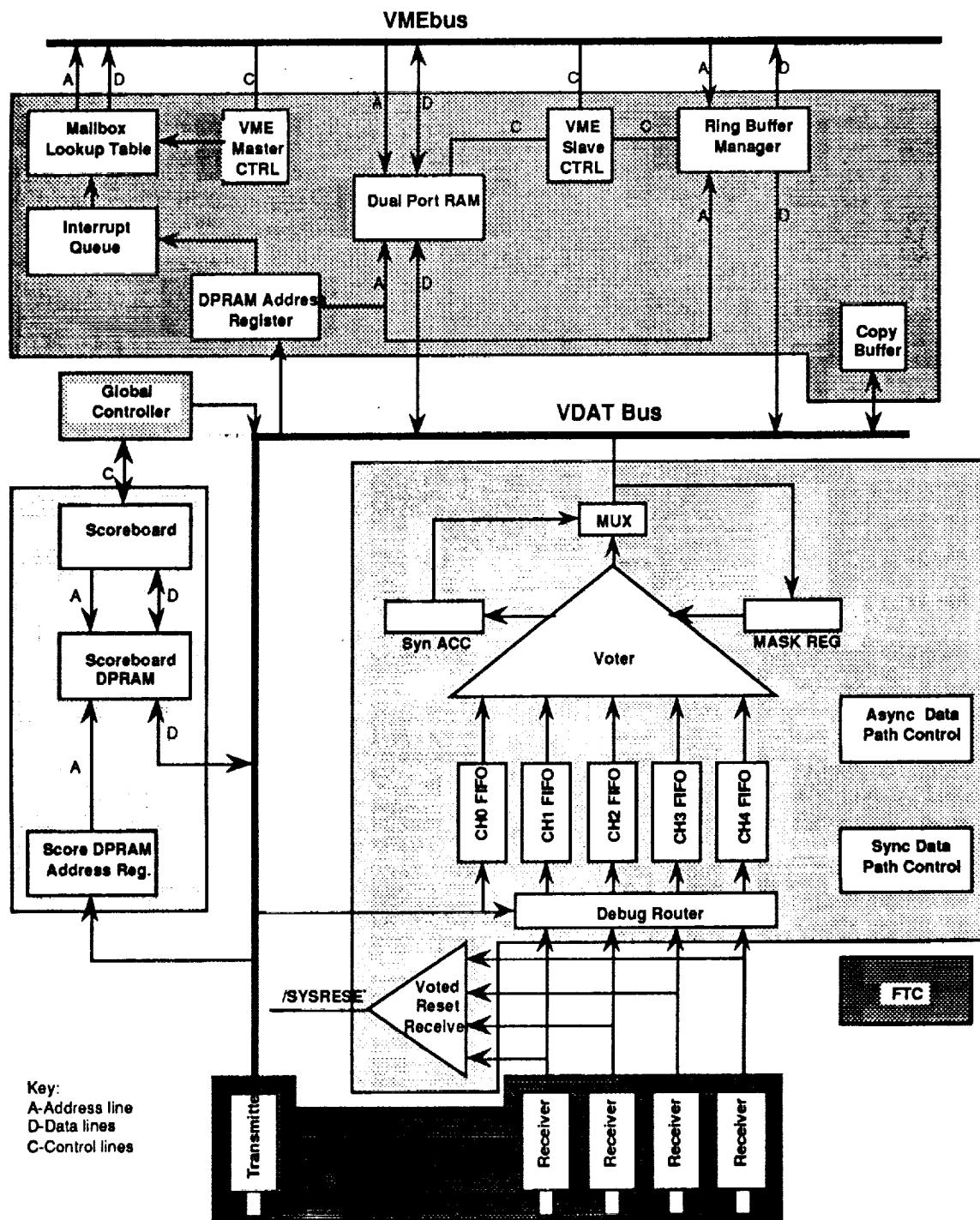


Figure 4-34. Functional Block Diagram of the AFTA Network Element

4.5.2.1.1. VMEbus Interface

The VMEbus interface connects the Network Element to the VMEbus in the backplane of the FCR. The NE is designed so that the VMEbus interface section contains the only

components that are specific to the VMEbus. The NE can be redesigned for a new type of backplane bus by simply replacing the VMEbus interface section with an interface to the new bus.

The interface to the VMEbus is further divided into two subsections. The slave interface is used to transfer data between the PEs and the NE and for accessing the ring buffer manager. The master interface is used to deliver mailbox interrupts to the PEs.

The slave section includes a dual-port buffer RAM for containing packet data either to be exchanged by the NE, or to be delivered to a PE. Since the buffer memory is a dual-port device, both sides may access the device simultaneously as long as they don't both access the same location. The ring buffer manager prevents such contention by assigning ownership to either the PE or the NE on a buffer cell by buffer cell basis. The PE and NE must be designed to adhere to the ownership specified by the ring buffer manager; there is no hardware enforcement of the ownership rules.

The master interface delivers mailbox interrupts to PEs. A mailbox interrupt can be delivered on either a packet transmission, packet reception, or input buffer full condition. Selection of the interrupt condition is done by modifications to the microcode in the global controller. Most PEs have mailbox interrupt capabilities located in the short address space (A16) of the VMEbus. If a particular PE for the AFTA does not have mailbox interrupt capabilities, or the capabilities do not meet the requirements expected by the NE mailbox interrupt delivery mechanism, that PE can not take advantage of the mailbox interrupt. However, it must be noted that the mailbox interrupt capability is optional; a PE can determine the same information by polling on the ring buffer manager. The purpose of the mailbox interrupt is to minimize, if possible, the amount of polling required.

4.5.2.1.2. Network Element Data Paths

The data paths of the NE perform the necessary data exchange patterns to correctly exchange, vote, and deliver data in the presence of faults. The data paths consist of a voter, synchronization FIFOs, controllers for the asynchronous and synchronous portions of the data paths, and a transmitter/receiver pair for voted resets and monitor interlocks.

The voter handles the resolution of multiple copies of data into a single copy. The voter has five inputs; however only one, three, or four copies are voted at a time. Voting of one copy simply involves delivering that copy to the output. The voting of three or four copies

requires a bitwise vote of each bit in the redundant copies. The voted result will always be correct if at most one of the inputs is faulty, unless simplex data is voted.

The selection of inputs to the voter is performed by the vote mask. The vote mask contains a number of registers, including the PE mask register, the NE mask register, and the source mask register. These registers are used in various combinations depending on what kind of packet exchange primitive is being executed. The loading of the vote mask registers and mask selection are done by the global controller.

The inputs to the voter come from a bank of five first-in/first-out (FIFO) devices. One FIFO is associated with each FCR in the system. Reference to a FIFO is made using the relative addressing mode. The purpose of the FIFOs is to synchronize the data coming into the NE. The NEs are synchronized to within a predetermined skew; however this skew is non-zero. Thus, data may arrive from each FCR at slightly different times. The FIFOs are used to buffer data as it arrives at the NE. The NE uses its internal synchronization reference, the fault-tolerant clock, to determine when the data is expected to arrive. The data is not read from the FIFOs until the data is guaranteed to be present (unless faults are present.)

The data path controllers shift data into and out of the FIFOs. The asynchronous controller shifts data into the FIFOs for each of the remote NEs, FIFOs CH1-CH4. The shift-in signals for the asynchronous FIFOs are derived from the inter-FCR communication (IFC) system. The synchronous controller shifts data into the FIFO for the local NE, FIFO CH0 and also shifts data out of all FIFOs. The shift-in signal for the synchronous FIFO, and the shift-out signals for all FIFOs, are derived from signals emanating from the global controller (GC).

The final major section of the NE data paths is the voted reset transmitter and receiver. The voted reset transmitter sends a voted reset command to all other NEs when a voted reset primitive is executed by the NE. When the NE receives a voted reset command over the IFC, the command is delivered to the voted reset receiver. The receiver votes the command received from each FCR and selects one or more discrete signals to assert based on the voted result. The discrete signal is asserted for a single FTC cycle following the voting of the voted reset command. The signal can be used to reset an element within the FCR, or to assert a monitor interlock. The one cycle assertion is sufficient to reset most elements in an FCR. If an element requires a longer reset signal, or if the discrete signal is to be asserted indefinitely, the signal must be latched. The latch circuitry is not a part of the NE.

4.5.2.1.3. Inter-FCR Communication System

The Network Elements are interconnected by the inter-FCR communication (IFC) system. The IFC system includes a transmitter, a fiber-optic network (not shown in Figure 4-34), and a bank of receivers.

The transmitter section of the NE converts incoming bytes from the NE into a serial bit-stream. The data is encoded on the bit-stream using a 4B/5B code [AMD89b] to provide sufficient transitions to ensure proper operation of the clock recovery circuitry on the receiver. The bit-stream is then converted to an optical signal for transmission over the fiber-optic network.

The fiber-optic network that interconnects the FCRs within an AFTA cluster provides a high bandwidth, high isolation interconnection network. Each NE drives a single output fiber; this fiber goes to a splitter where the optical signal is replicated four times. The four outputs of the splitter are delivered, one to each NE (all except the original transmitter). The fiber-optic network contains a splitter for each NE. A diagram of the fiber-optic network is shown in Figure 4-35.

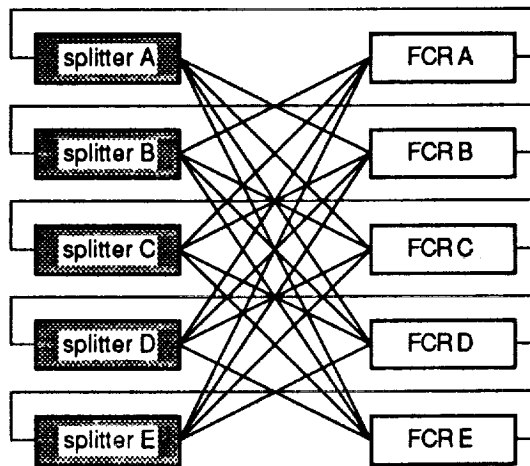


Figure 4-35. Inter-FCR Fiber-Optic Network

The data receivers on the NEs perform the inverse function of the transmitters. First, the optical signal from the fiber-optic network is converted to an electrical signal. The data clock is recovered from the serial signal and is used to convert the incoming signal back into an 8-bit wide data stream. The receiver asserts a signal, based on the data clock, to signal the asynchronous data path controller that the data on the output of the receiver is valid.

4.5.2.1.4. Fault-Tolerant Clock

The fault-tolerant clock (FTC) circuit is a free-running digital phase-locked loop. The FTC in each FCR tries to maintain synchronization with the perceived median FTC signal from the other FCRs. Adjustments are made by adding or deleting a single clock cycle from the normal FTC period. All adjustments are made during a known adjustment period; the NE is designed to tolerate one more or one fewer clock cycles during the adjustment period.

A signal called a bound is generated for each remote FCR from the IFC receivers. The four bound signals are voted by a median-edge voter to select the second observed edge. The voted signal, called median bound, is compared to the local FTC signal delayed by the expected inter-FCR delay (from empirical measurements). If the local signal is perceived to be ahead of the median bound, a self-ahead adjustment is made by adding a single clock cycle to the adjustment period. If the local signal is perceived to be behind the median bound, a self-behind adjustment is made by deleting a clock cycle from the adjustment period.

The Figures 4-36 through 4-38 demonstrate the three types of FTC adjustments. The rising edge of the median bound is compared to windows relative to the local FTC signal. If the rising edge occurs within the normal (N) window, no adjustment is made. If the edge occurs in the self-ahead (A) window, a self-ahead adjustment is made, and if the edge occurs in the self-behind (B) window, a self-behind adjustment is made. The error window (E) indicates that the local FTC is skewed too much from the median bound to be considered synchronized with the other FTCs. The rising edge of median bound should never occur within the error window except during initial synchronization of the FTC signals.

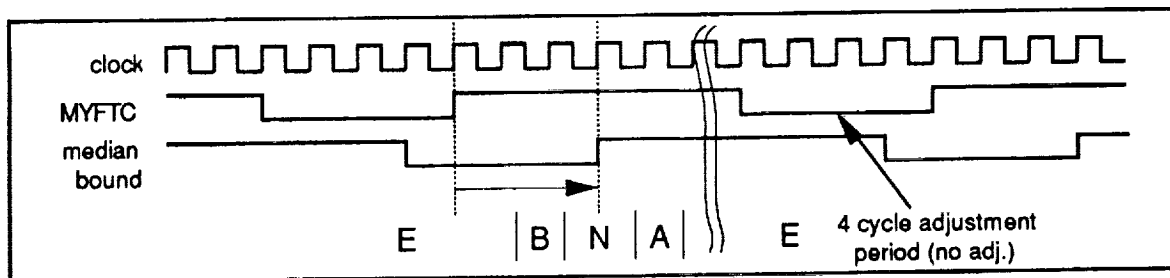


Figure 4-36. Normal FTC Adjustment Period

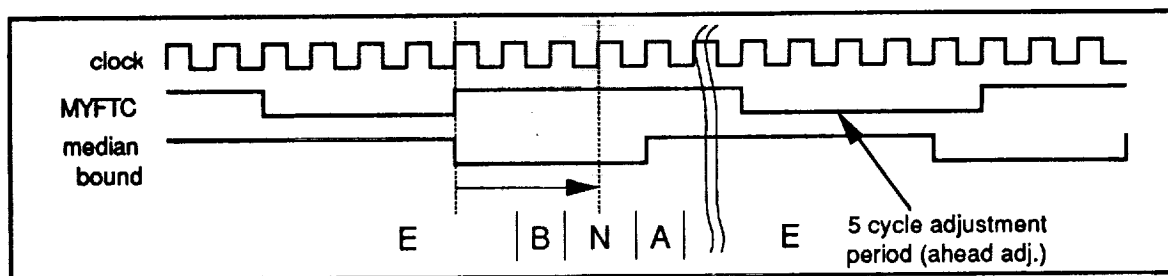


Figure 4-37. Self-Ahead FTC Adjustment Period

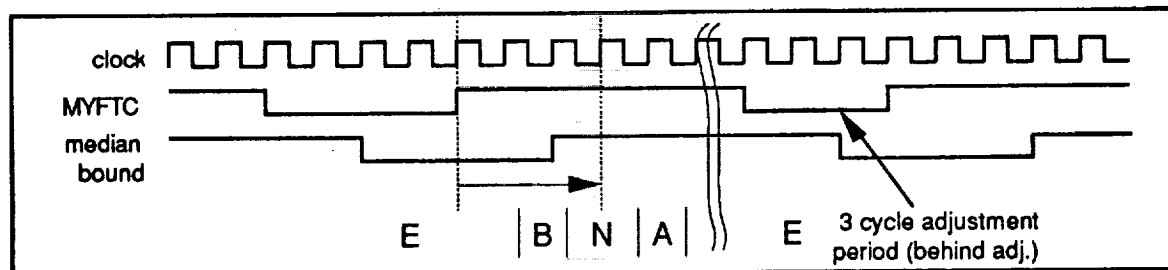


Figure 4-38. Self-Behind FTC Adjustment Period

4.5.2.1.5. Global Controller

The global controller coordinates the functions throughout the Network Element. The GC asserts signals in almost all other major sections of the NE. The GC is a microcoded finite-state machine. The GC also has the capability of driving constant data onto the VDAT bus; this capability is used to load initialization parameters into other sections of the NE.

The microcode store for the GC is built out of registered RAMs with a serial scan-path for initialization. The microcode is easily changed by creating a new load module which is transferred to one of the PEs during the booting process. The PE responsible for initializing the NE transfers the microcode to the GC before proceeding with self-tests or ISYNC.

An embedded system has no need for a flexible microcode store. Indeed, it may be more desirable from a reliability standpoint to use non-volatile storage. Thus, the GC is designed so that registered PROMs can be easily substituted for the registered RAMs.

4.5.2.1.6. Scoreboard

The scoreboard is the key element in the AFTA. The scoreboard is responsible for approving the execution of exchange primitives in a manner consistent with Byzantine resilience.

The NEs periodically perform a poll of the buffer status of each physical processor in the AFTA. The status includes whether the processor has room in its input buffers to contain a packet (the input buffer not full, or IBNF, condition) and whether it has a packet in its output buffers to be exchanged (the output buffer not empty, or OBNE, condition). If the latter is true, the exchange primitive to be executed, the destination of the packet, and a user-defined byte are also included.

The aggregate of buffer status polls is called the system exchange request pattern, or SERP. The SERPs are exchanged such that each functioning NE is guaranteed to have a SERP copy that agrees with all other SERP copies; thus, each NE makes decisions based on the SERP with confidence that the other NEs will make the same decision.

The exchanged SERP is delivered to the scoreboard for processing. The scoreboard uses a virtual group to physical processor mapping to extract the buffer status information from the SERP on a VG-by-VG basis. The individual buffer status bits contained in the SERP for the virtual group members are voted to determine the overall status of the VG. If all members of a VG, a unanimity, assert a status bit, the condition is considered true. If a majority, but not a unanimity, of VG members assert a status bit, the condition is considered almost true, and a timeout is started. If the timeout expires before the remaining members assert the status bit, the condition becomes true anyway. In any other situation, the condition is considered not true.

When the OBNE condition is observed true for a VG, that VG becomes a packet source. The class, destination VID, and user byte fields for the source are voted to determine the requested packet exchange primitive to be executed and the destination of the packet. The voted destination VID field is used to look up the status of the IBNF for the destination VG. If the IBNF condition is not true, the destination's input buffers are full, causing flow control to be asserted. If the IBNF condition for the destination VG is true, the destination has room to receive at least one packet. In the latter case, the scoreboard specifies the NE to execute the requested exchange primitive as determined by the class field, and deliver the resulting packet to the virtual group specified by the destination VID field. The contents of the user byte are voted (but not used) by the scoreboard and delivered to the destination.

4.5.2.2. Network Element Physical Characteristics

The following table outlines the characteristics of the AFTA brassboard Network Element. Since the NE is currently only a conceptual design, these characteristics are estimates derived from experience and preliminary design parameters.

Weight	1.5 pounds (estimated)		
Power	7A@5VDC		
Volume	Height	9.187 inches	
	Depth	6.299 inches	
	Thickness	0.063 inches (board)	
		0.800 inches (front panel)	
Failure rate	See Section 9.		
Device technology	TTL/CMOS		
Operating Temperature Range	0 to 55°C at inlet to cooling fans		
Storage Temperature Range	-40 to +85°C		
Relative humidity	5% to 90%, non-condensing		
Cooling Requirements	10 SCFM air flow over nominal operating temperature range		

Table 4-4. Characteristics of AFTA Network Element

4.5.2.2.1. Circuit Board Layout

Figure 4-39 shows a preliminary board layout for the AFTA brassboard Network Element.

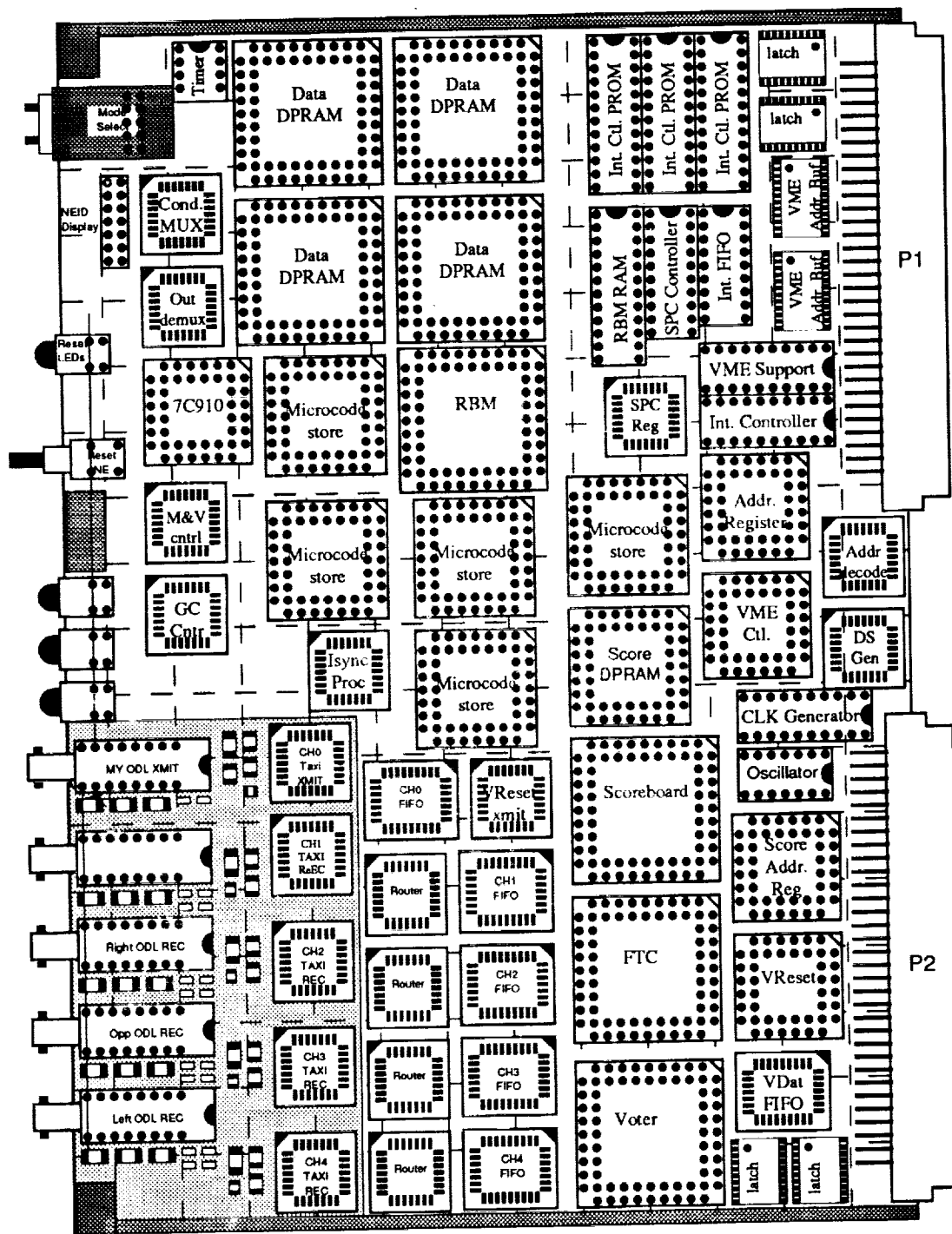


Figure 4-39. Network Element Brassboard Layout

4.5.2.2.2. *Military Qualification of Baseline Network Element*

The AFTA is targeted for application in hostile military embedded environments. Thus, the availability of the components of the AFTA in military-qualified versions is an important factor in the acceptance of the AFTA for such applications. Since the NE is the only hardware specific to the AFTA, the availability of the individual integrated circuits of the NE in military-qualified form is necessary.

Throughout the conceptual design, two major military publications, MIL-STD-883C and MIL-M-38510, were used as criteria for the qualification level of components for the AFTA NE. The goal is to ensure that all backplane-independent components (i.e. anything not in the VMEbus interface subsection of the NE) are available qualified to MIL-STD-883C, Class B and specified by either a MIL-M-38510 "slash sheet" or on a standard military drawing (SMD). Alternatively, equivalent functionality in a similar part is desired. Table 4-5 summarizes the military availability of devices for the AFTA.

Many of the parts do not have exact equivalents in military-qualified versions. However, with a few exceptions, the NE functionality can be captured in devices that are very similar to those used in the brassboard. The reason the brassboard does not use these similar parts in the first place is due to space, performance, and cost considerations. The following paragraphs summarize the status of the parts which do not have exact military-qualified equivalents.

The status of the scoreboard is indeterminate, since at this time, the design and implementation of the scoreboard has not been finalized. However, a "Baseline Network Element" has been identified which consists of the NDI devices listed in the Table below and a single Application-Specific Integrated Circuit (ASIC) which implements the Scoreboard functionality. It is believed that this level of integration of the Scoreboard is necessary for the NE to fit onto a single MIL-STD-344 LRM. Subsequent calculations of AFTA failure rate, weight, power consumption, and volume will refer to this version of the NE.

Manufacturer	part	Mil-Std	SMD Number
CSDL	scoreboard	n/a	n/a
IDT	7202	883B(C?)	5962-8753101 (7201LA30) ³ 5962-8866904 (7203S40) ³
	6116	883B(C?)	5962-8874002 (6116LA25) ²
	7134	883B(C?)	5962-8700201 (7132SA-45) ³ 5962-8700205 (7132LA-45) ³
	72402	883B(C?)	5962-8684604 (72404L35) ³ 5962-8684603 (72404-25) ³
	7006	883B(C?)	n/a
	39C10	883B(C?)	5962-8770803 (39C10C) ⁴
	71502	883B(C?)	n/a
Signetics	SCB68172	n/a	5962-8770501
Altera	EPM5032	883B(C?)	5962-90611 (CY7C344) ¹
	EPM5064	883B(C?)	n/a
	EPM5128	883B(C?)	5962-89468 (CY7C342) ¹
	EP910	883B(C?)	
Lattice	GAL16V8	883C	5962-89839032A
	GAL18V10	883C	n/a
	GAL22V10	883C	5962-8984103LA
			5962-8867001 (C22V10, Wind.) ¹
			5962-8872401 (C22V10, Opaq.) ¹ 5962-8872401 (C22V10L, Opaq.) ¹
TI	GAL26V12	883C	n/a
	ALS245	54ALS	5962-8403001
AMD	ALS646	54ALS	5962-8995601
	7968	883C	DH27023 (LCC Temp Waiver) DH27025 (LCC Voltage Waiver)
Dallas	7969	883C	DH27024 (LCC Temp Waiver) DH27026 (LCC Voltage Waiver)
	1232	n/a	n/a
Cypress	7C245A	883C	5962-89815 (Wind., pnd. June) 5962-88735 (Opaque)
AT&T	ODL Xmit	883C	
	ODL Receive	883C	

Notes :

¹ Second source part.

² Speed variation or technology variation from specified part.

³ Similar part to specified part.

⁴ Different or specified revision of part.

Table 4-5. Military Device Availability for Network Element

Several devices from IDT are not available in a military-qualified form. Some, such as the FIFOs and the small dual-port RAMs, have similar devices with less internal memory. For the FIFOs, this is no problem. For the DPRAMs, more devices can be used in parallel to provide the same amount of memory. Other devices, including the large dual-port RAMs

(IDT 7006) and the registered RAMs (IDT 71502), are not available in any similar part. However, the large DPRAMs are part of the VMEbus-specific section, and the registered RAMs can be easily replaced with Cypress registered PROMs (CY 7C245A), which are available in an SMD form. The functionality of the registered PROMs and the registered RAMs is very similar, as is their timing characteristics. The GC is designed to function with either of these two forms of control store.

The Altera EPM5064 is not currently available in an SMD form, although it is qualified to MIL-STD-883B. However, the complete functionality of the EPM5064 can be captured in an EPM5128, which is available in an SMD form. Since the EPM5128 comes in a larger package, board space must be sacrificed to accommodate EPM5064 designs in an EPM5128.

The Lattice 22V10 is a very common PAL architecture and is available in an SMD form from many vendors, including Lattice. The 18V10 and 26CV12, variations to the 22V10 architecture, are not available on the SMD list. However, the functionality of an 18V10 can be implemented in a 22V10, and the functionality of a 26CV12 can be implemented in two 22V10s with the sacrifice of board space.

The AMD 7968 and 7969 devices, the TAXI transmitter/receiver pair, are only available in a waived form, either temperature or voltage waived. While waived parts are to be avoided, these are the only devices that perform the necessary function.

The Dallas device is not available in a military-qualified form. However, this is not a problem since the function of the Dallas device, a watchdog timer, is not essential to the AFTA functionality. If the functionality of a watchdog device is desired (which it probably is, for common-mode fault recovery), the same functionality can be implemented using several discrete components with the sacrifice of board space.

The AT&T devices are the fiber optic data links that convert between electrical and optical signals. Since they are hybrid devices, they are not covered by the SMD list. However, surface-mount versions of the device, qualified to MIL-STD-883C, have been announced by AT&T [APS90].

4.5.2.3. Effect of Implementation Technology on Network Element Physical Characteristics

The technology used to implement the NE determines its performance, power consumption, weight, volume, and failure rate. The Baseline design outlined above uses one ASIC to implement the Scoreboard functionality. Because of the cost involved in fabricating an ASIC, two additional implementation options for the Scoreboard have been considered for the NE. The first relates to the implementation of the Scoreboard function using a high speed Reduced Instruction Set Computer (RISC) processor and the second relates to the Scoreboard implementation using a number of Field Programmable Gate Arrays (FPGAs). For the reasons listed below, neither of these implementation options appears sufficiently attractive to warrant their continued development.

4.5.2.3.1. RISC Processor Scoreboard

Presented below is a listing of the pin counts, total gates, and power consumption of a the AFTA Network Element Scoreboard implemented as an AMD29000 processor. This information is intended for comparison purposes only, not as any recommendation for this course of action. Below is also a paragraph explaining some of the penalties associated with this approach.

The main drawback to this approach is the massive performance penalty. The code which gets executed most often is the voting code. Using optimized assembly language, this code comes out to 99 instructions. This section is executed at least once for every VG in the system. Assuming an all triplex configuration (13 VGs) this means 99 total instructions. Using a 40MHz processor (25ns per instruction), the scoreboard emulator will take 99 μ s just to vote the OBNE bits of the VGs. When the overhead of performing timeouts and voting the remainder of the SERP information is added, the scoreboard emulator will be too slow to support real-time tasks with iteration rates of 100Hz. Thus, this method will not meet the performance requirements of the AFTA.

The total gate count provided is intended to convey the number of transistors present rather than a size estimate. For this purpose the RAM counts are doubled. This is due to the fact that two 'gates' are used for each RAM cell. A gate is defined as four transistors (two N-channel and two P-channel).

ScoreBoard Emulator

AMD29000 RISC Processor		
Power Consumption (Watts)	~4.125	
Pin Count	169	
Dual Port RAM (Jdt 7134 or equivalent) [4K x 8]		
Power Consumption (Watts)	1.5	
Pin Count	48	
Gate Count	70K	
Processor RAM (Jdt 7186 or equivalent) 2x[4K x 16]		
Power Consumption (Watts)	1.5	
Pin Count	88	
Gate Count	262K	
Miscellaneous Glue Logic (Altera 5064 or equivalent)		
Power Consumption (Watts)	1.5	
Pin Count	44	
Gate Count	5K	
<u>Functional Scoreboard Emulator totals</u>		
Power Consumption (Watts)	8.625	
Pin Count	349	
Gate Count	337K	+ Processor complexity

4.5.2.3.2. *FPGA Implementation*

Presented below is a listing of the pin counts, total gates, and power consumption of an AFTA Network Element Scoreboard implemented in FPGAs. This information is intended for comparison purposes only, not as any recommendation for this course of action. Below is also a paragraph explaining some of the penalties associated with this approach.

There are three major reasons for not taking this approach. The first is the complexity. A student at CSDL recently completed two FPGA designs for his thesis, one of which was a voter. The voter consumed an entire FPGA by itself, and the scoreboard contains a voter plus other custom hardware. The second reason is that the existing VHDL models would be invalid for the new design. Some companies have promised VHDL support for their FPGA design systems, such a capability is at least a year and a half away. With all the effort put into the VHDL modeling, it would be wasteful to throw it all away. Closely coupled to this is the third reason, verification. With the VHDL model the verification occurs with the transformation to the gate level. Each new step could be verified before continuing. With the FPGA approach, each FPGA would be a segment of the scoreboard algorithm, causing problems with verifying it individually.

The total gate count provided is intended to convey the number of transistors present rather than a size estimate. For this purpose the RAM counts are doubled. This is due to the fact that two 'gates' are used for each RAM cell. A gate is defined as four transistors (two N-channel and two P-channel).

ScoreBoard Emulator (FPGA)

2 Dual Port RAMs (Jdt 7134 or equivalent) [4K x 8]	
Power Consumption (Watts)	3.0
Pin Count	96
Gate Count	140K

4 Altera 5128 FGAs	
Power Consumption (Watts)	5.0
Pin Count	272
Gate Count	32K

<u>Functional Scoreboard Emulator totals</u>	
Power Consumption (Watts)	8.0
Pin Count	368
Gate Count	172K

4.5.2.3.3. *High-End Network Element*

A third implementation study performed under the Conceptual Study relates to the aggressive use of VHSIC/VLSI packaging to construct the Scoreboard, Global Controller, Voter, and VMEbus Controller in four ASICs. This design is denoted the "High End Network Element."

Presented below is a listing of the pin counts, total gates, and power consumption of an AFTA Network Element consisting of four ASICs and 16K x 32bits of DPRAM. The DPRAM was not included into the ASIC design since it was so large.

Each ASIC is shown below followed by its clock speed. Shown below the name are the parts which were used to estimate the gate counts. When there is a number in square brackets, [], it indicates a part which would have been used had the ASIC approach not been taken. Following this number is a calculation showing how the equivalent number of gates was arrived at. For example, the FTC section of the Voter ASIC contains the following : FTC [5128] (256x48x0.8). This means that the FTC would have been implemented in an Altera 5128, containing 256 internal Macro cells. Each Macro cell is estimated to contain 48 gates of functionality and the chip is considered to be at 80% utilization. Thus 9.8K

gates is reached. Any multiplier before the chip type is the number which would have been used. For RAM the information is the number of chips used, their depth, and their width.

The total gate count provided in the summary is intended to convey the number of transistors present rather than a size estimate. For this purpose the RAM count was doubled and added to the logic count. This is due to the fact that two 'gates' are used for each RAM cell. A gate is defined as four transistors (two N and two P-channel). For a size estimate, the RAM cells take about 43 percent of the space required for the logic cells, so the original RAM count should be multiplied by 0.43 and added to the logic count to receive a total 'soft gates' equivalent. Since the voter has minimal RAM requirements it may be implemented in an LSI LCA package without using separately diffused RAM.

Any errors in the data below should be on the conservative side. All calculations presented were done in the most conservative manner possible. One assumption made was that 25% of the gates could be active on any clock edge. LSI has stated that typically only 15-20% will be active. Another was the total number of gates used for RAM has been included in the gate count. Most likely the RAM usage will come from the raw gate count due to its being diffused into the silicon directly and not being implemented in the 'soft gates' the logic will use. It will therefore take less space than the 'soft gates' and not impact as much of the useable area on the die.

ScoreBoard (25Mhz)

	<u>Logic</u>	<u>RAM</u>
Random Logic	50K	
RAM		40K
	-----	-----
Totals	50K	40K

Logic Gate Power consumption : (25 % x 5.5 μ W/gate/MHz)

$$0.25 \times 5.5 \times 10^{-6} \times 50K \times 25 = 1.719 \text{ Watts}$$

Pin Count

Data	(4x32)	128
Address		10
Control		7

Total		145

Pin Power consumption : (25 μ W/pin/MHz/pF)

$$25 \times 10^{-6} \times 145 \times 25 \times 5 = 0.453 \text{ Watts}$$

Summary:

Pin Count :	145
Total Gates :	130,000
Total Power (Watts):	2.172

<u>Global Controller</u> (12.5Mhz)		<u>Logic</u>	<u>RAM</u>
Controller		2K	
Misc Logic (Mux, etc.)		0.5K	
Control Store			328K
		-----	-----
Totals		2.5K	328K

Logic Gate Power consumption : (25 % x 5.5 μ W/gate/MHz)

$$0.25 \times 5.5 \times 10^{-6} \times 0.5K \times 25 = .05 \text{ Watts} + .5 \text{ Watts} = .55 \text{ Watts}$$

Pin Count

Control 80

Pin Power consumption : (25 μ W/pin/MHz/pF)

$$25 \times 10^{-6} \times 80 \times 25 \times 5 = 0.125 \text{ Watts}$$

Summary :

Pin Count : 80

Total Gates : 657,860

Total Power (Watts): 0.675

<u>Voter</u> (12.5Mhz)	<u>Logic</u>	<u>RAM</u>
FIFOs (5x64x32)	0.5K	10K
Debug Router (4x32x2:1)	0.3K	
VDAF FIFO (64x32)	0.1K	2K
Voter [5128] (256x48x0.8)	9.8K	
FTC [5128] (256x48x0.8)	9.8K	
VReset REC [5064] (128x48x0.8)	4.9K	
VReset XMIT [5032] (64x48x0.8)	2.5K	
Isync Proc. [5032] (64x48x0.8)	2.5K	
	-----	-----
Totals	30.4K	12K

Logic Gate Power consumption : (25 % x 5.5 μ W/gate/MHz)

$$0.25 \times 5.5 \times 10^{-6} \times 30.4K \times 25 = 0.142 \text{ Watts}$$

Pin Count

Data	(5x32)	160
FTC		5
DS		5
Clock		1
Buffer Cntl.		6
M/I		16

Total		193

Pin Power consumption : (25 μ W/pin/MHz/pF)

$$25 \times 10^{-6} \times 193 \times 25 \times 5 = 0.302 \text{ Watts}$$

Summary :

Pin Count :	193
Total Gates :	54,976
Total Power (Watts):	1.37

VME Controller (25Mhz)**Logic****RAM**

VME Buffer 4x[245]	0.4K	
VME Addr. Buffer 2x[646]	1.0K	
VME DS gen [26V12]	0.8K	
VME Ctlr. [5064] (128x48x0.8)	4.9K	
VME Addr. Decode [22V10]	0.5K	
DPRAM Addr. Reg. [5064]	4.9K	
INT FIFO (64x5)	1.5K	0.3K
INT Ctl. PROM 3x[7C245] (2048 x 8)	0.1K	49K
RBM [5128] (256x48x0.8)	9.8K	
RBM RAM [6116] (2048x8)		16.4K
	-----	-----
Totals	23.9K	65.9K

Logic Gate Power consumption : (25 % x 5.5 μ W/gate/MHz)

$$0.25 \times 5.5 \times 10^{-6} \times 23.9K \times 25 = 0.821 \text{ Watts}$$

Pin Count

Data (2x32)	64
Address	46
INT & Control	21

Total	131

Pin Power consumption : (25 μ W/pin/MHz/pF)

$$25 \times 10^{-6} \times 131 \times 25 \times 5 = 0.205 \text{ Watts}$$

Summary :

Pin Count :	131
Total Gates :	155,612
Total Power (Watts):	1.026

Dual Port RAM

VME DPRAM (idt 7006 or equivalent) 4x[16K x 8]

Summary :

Pin Count :	272
Total Gates :	1,048,576
Total Power (Watts):	2.5

4.5.3. Input/Output Controller (IOC) Characteristics

A host of mission-specific input and output devices are connected to the AFTA by input/output controllers (IOCs). An IOC is an LRM that contains one or more I/O devices. Examples of IOCs include interfaces to IMUs, GPS receivers, CNI gear, displays, status

panels, AHARS, air data sensors, altitude, image, range, Doppler navigation, engine sensors and controls, switches and annunciators, and data buses such as MIL-STD-1553.

A number of IOC types may reside in the AFTA; an IOC plugs into the FCR backplane. PEs communicate with the IOCs either over the FCR backplane bus or over a dedicated I/O bus; the advantage of the latter approach is that the I/O traffic does not interfere with the inter-VG traffic between the PEs and the NEs, which goes over the FCR backplane bus. The exact definition of the I/O suite is mission-specific. The AFTA architecture is designed to admit any IOC that is compatible with the selected backplane bus. The IOC may simply be a device-specific memory-mapped controller, or it may be an interface to a dedicated I/O network.

“Dumb” IOCs are controlled by a PE (either a simplex VG or one member of a redundant VG) and never communicate directly with the NE. “Smart” IOCs, for all intents and purposes, act like PEs and may be grouped into redundant VGs; dumb IOCs can not.

The following I/O devices are candidates for implementation in the AFTA; this list will be refined as detailed data regarding the mission’s requisite I/O suite become available.

- Instrumentation buses: MIL-STD-1553/1773 (SAVA 1553M)

- LANs: FDDI, SAFENET II, Ethernet

- Memory-mapped I/O: A/D, D/A, discretes, SAVA ADM

- Expansion memory: non-volatile RAM, EEPROM

- Mass memory: disk, tape, CD-ROM, SAVA MM

- Fault Tolerant Data Bus/Authentication Module

4.5.4. Power Conditioner (PC) Characteristics

The power conditioners (PCs) supply the power required by the AFTA. At least one PC is used for each AFTA FCR; to eliminate single-point failures, a PC can not be allowed to drive more than one FCR. The PCs not only regulate and filter the power delivered to an AFTA FCR, they also maintain uninterrupted AFTA operation in the presence of momentary dropouts on the main vehicle power buses. A simplified architecture of a PC is shown in Figure 4-40.

A PC provides regulated power conversion from the main vehicle direct-current (DC) power buses, labeled “A” and “B” in Figure 4-40, to the voltage level(s) necessary to power the AFTA LRMs. One or more sense signals are fed back from designated points in

the FCR to allow closed-loop feedback of the regulated power as FCR load varies. The PC may contain multiple regulators for various voltage levels or to power isolated sections of the FCR. Optional PC capabilities, not shown in Figure 4-40, include selective activation and deactivation of a PC and remote monitoring of the PC output voltage and current.

To tolerate the indefinite loss of either of the two main vehicle buses, each PC is connected to both main buses. Diodes CR1 and CR2 prevent current from flowing from an operational to an inactive main bus. The PC is designed to provide full rated power to the FCR using only one bus; thus the AFTA can be used in vehicles which possess only one main power bus. The PC must also tolerate the temporary outage of both buses, such as during the switch-over from helicopter APU to engine generators, during which a 0.5s power dropout occurs on both vehicle buses. This function is provided by a battery in each PC which must be sized to provide full FCR power for the duration of any anticipated outage of both vehicle main buses. The PC must also protect itself and its FCR from over-voltages on the main buses; this protection is provided by the units labeled OV1 and OV2 in the figure. Metal-oxide varistors (MOVs) are included in the OV1 and OV2 circuitry to handle high-frequency high-voltage input spikes which are too fast for conventional input overvoltage protection circuitry. Finally, fuses (or circuit breakers), labeled F1 and F2 in Figure 4-40, limit input current from either main vehicle bus. These fuses must be sized to permit power-on current surges in excess of the expected steady-state power dissipation.

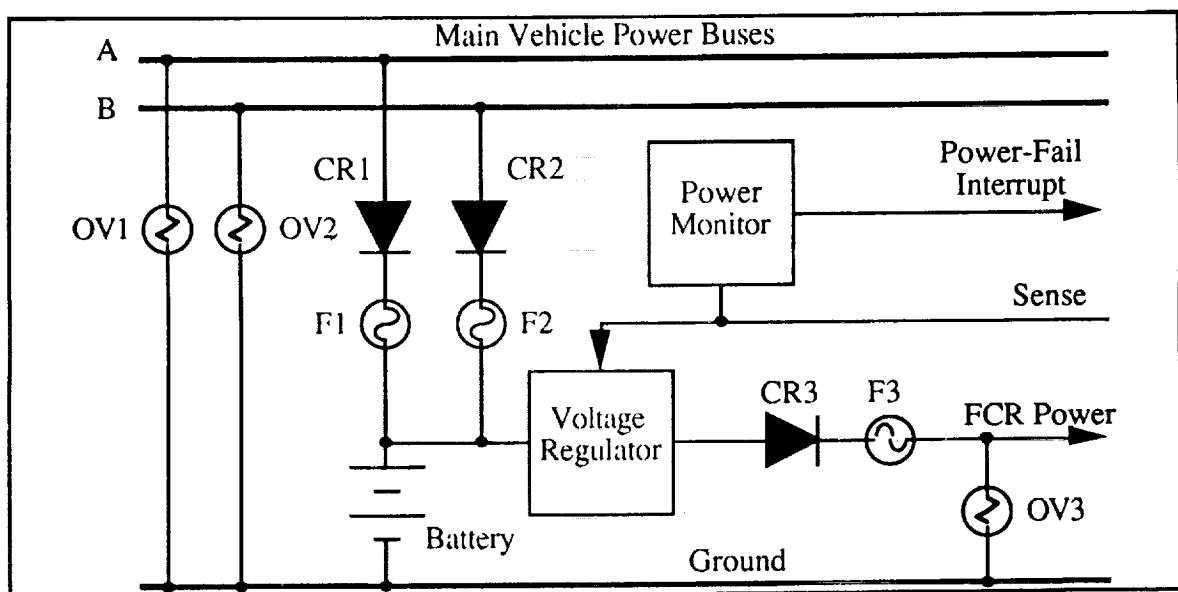


Figure 4-40. Functional Block Diagram of the AFTA Power Conditioner

The FCR is protected from excessively high PC output voltage and high-voltage high-frequency spikes by the output overvoltage protector labeled OV3; this circuitry also protects the PC and main vehicle buses from back-propagated overvoltages and spikes emanating from the FCR. Diode CR3 prevents back currents generated within the FCR from flowing into the PC and possibly onto the vehicle main buses. The fuse (or circuit breaker) F3 provides current-limiting protection for the FCR. This fuse must be designed to permit the expected power-on current surge.

If an output undervoltage condition persists for a time interval which exceeds the battery's amp-hour rating, or if the voltage regulator itself fails, the PC's power monitor asserts a power-fail interrupt to the FCR to allow the FCR to attempt emergency power-down functions before the PC output drops below a minimal level. Since the AFTA is Byzantine resilient, the sudden loss of any single FCR to an undervoltage condition or PC failure is tolerated; however, the power fail interrupt provides a mechanism for a more graceful degradation of the AFTA system.

4.5.5. Cooling System

All AFTA components utilize a thermal management layer for conductive cooling from the LRM interior to the LRM edge. The edges of each LRM are thermally connected to the LRU side walls using wedge locks. The LRU side walls are in thermal contact with the LRU cold plate, which transfers heat away from the LRU. Depending on the installation, the LRU cold plate may be cooled by ambient air, forced air, or forced liquid.

5. AFTA Software Architecture

5.1. Overview

The AFTA is designed to provide a highly reliable parallel processing system. Processing is distributed among the parallel processing sites by task, and intertask communication is provided by message passing. High reliability is provided by redundantly executing the tasks on replicated processors. The AFTA hardware and software have been designed to hide the hardware redundancy, hardware faults, and the parallel processing details from the applications programmer. The functional structure of the system software is shown in Figure 5-1. Each VG in the system executes the rate group tasking paradigm which provides the execution environment for application and system service tasks. FDIR is a system service which provides fault detection, isolation, and reconfiguration. The I/O services provide the interface to input/output devices.

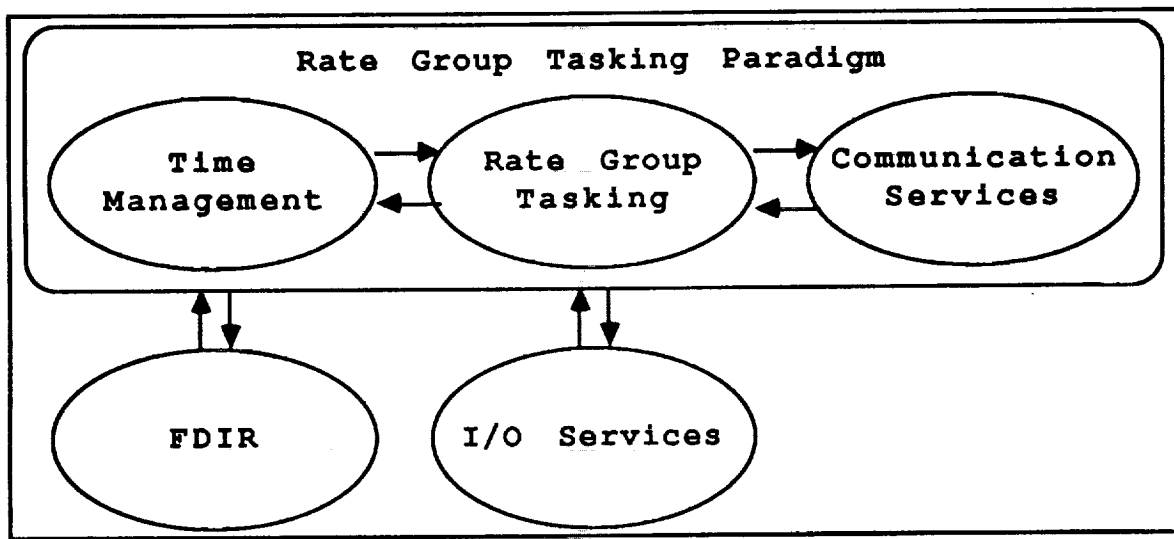


Figure 5-1. AFTA System Software Organization

A desired initial system configuration must be specified prior to beginning system operation. The configuration must specify the mapping between tasks and VGs and that between VGs and processors. This mapping is maintained by the operating system and is used to isolate the applications programmer from the underlying redundancy and parallel processing mapping. System initialization uses the above mapping to test and ready the hardware components of the system and evaluate whether there are sufficient resources to perform the mission. System initialization must also ready the software components for execution. The system specification and an overview of system initialization is described in Section 5.2.

The rate group tasking paradigm provides the framework for executing system and application tasks in the AFTA. It is composed of the rate group tasking services, the time management services, and the communication services. The rate group tasking services define the tasking framework for the application and system service tasks and are described in Section 5.3. The time management services are used to generate the rate group frame boundaries and provide periodic resynchronization of the VG. They are described in Section 5.4. The communication services are used for intertask communication and are described in Section 5.5.

FDIR provides fault detection, isolation, and reconfiguration of Processing Elements in the system. It is composed of local FDIR which executes on each VG and system FDIR which executes on a specially designated VG. Local FDIR has the responsibility for detecting and isolating hardware faults in the processor elements of its VG and disabling their outputs using the interlock hardware. In addition, local FDIR reports all link and Network Element (NE) faults to system FDIR and responds to its reconfiguration commands. It is also responsible for transient PE hardware fault detection and for running low priority PE self tests to detect latent PE faults. It is transparent, autonomous and non-intrusive to the application. The system fault detection, isolation and reconfiguration is responsible for the collection of status from the local FDIR and detection, isolation and masking of Network Element faults, and link faults. It resolves conflicting local fault isolation decisions, isolates unresolved faults, correlates transient faults, and handles VG failures. Local and system FDIR are both described in Section 5.6.

The I/O services provide efficient and reliable communication between the application program and external devices (sensors and actuators). They execute on any VG which is responsible for I/O and provide source congruency on all input data and voting of all output data. The I/O services provide the user with the ability to group I/O transactions into chains and I/O requests. It also provides the user the flexibility of scheduling both preemptive and non-preemptive I/O. The I/O services are described in Section 5.7.

5.2. System Specification and Initialization

The AFTA is composed of a set of processing and Network Elements. During system initialization these components are individually initialized and tested. Non-faulty components are then grouped into redundant units which can provide fault tolerant operation. The Processing Elements are grouped into a set of virtual groups or VGs and the network elements are grouped into the Network Element aggregate. These units are then initialized and tested. System initialization is completed by distributing the application load among the available VGs and beginning cooperative execution of the assigned tasks.

The initialization is based on a VG configuration table and a task configuration table. The VG configuration table maps VGs to Processing Elements. The task configuration table maps rate group tasks to VGs. An initial configuration must be specified for both tables and compiled into the AFTA operating system. The VG configuration table and associated initialization are discussed in Section 5.2.1. The task configuration table and associated initialization are discussed in Section 5.2.2. Detailed descriptions of the initialization procedures are discussed in their associated sections.

5.2.1. Virtual Group Configuration

The Network Elements provide message passing between VGs. To provide the communication, the Network Elements must map each VG to its corresponding set of Processing Elements. This mapping is maintained in the Network Element's VG configuration table. The VGs also maintain a copy of the configuration table to alter the VG to processing element mapping when operational requirements or resource availability changes. The processing element is specified by its hosting Network Element and the Network Element port it is using for communication. During initialization, the processors contend for the available communication ports and the above mapping does not provide a unique external physical identification of the corresponding processor hardware. The physical identifier of each processor in a VG is maintained in the software version of the table for identification of faulty elements. An example AFTA configuration showing the information used in the configuration table is shown in Figure 5-2.

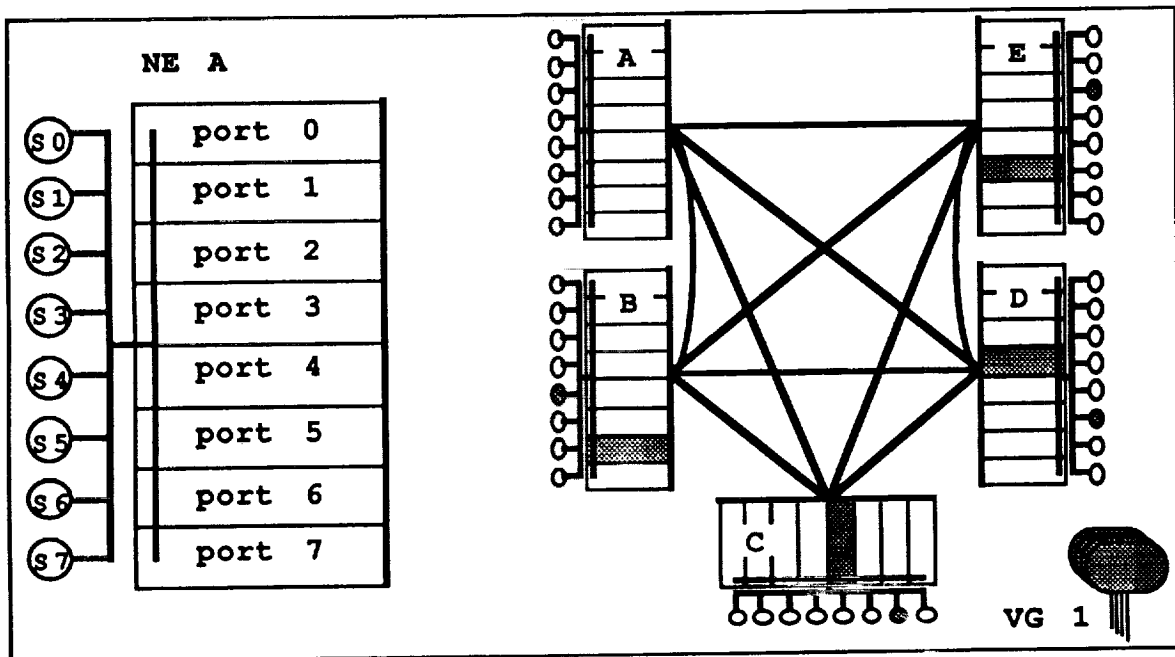


Figure 5-2. Example AFTA Configuration

The example system has five Network Elements with each Network Element having eight communication ports and hosting eight Processing Elements. VG 1 is a quadruplex composed of Processing Elements (B,6), (C,4), (D,3), and (E,5) where the first element of the pair is the Processing Element's Network Element id and the second element is its port id. This is the information required by the Network Elements for delivery of the communication service. The OS also maintains the physical identifier of each processor in the VG to externally identify faulty hardware. The corresponding physical identifiers of the processors in VG 1 are (B,S4), (C,S6), (D,S3), and (E,S2) where the first element of the pair is the Processing Element's Network Element id and the second element is an externally visible identifier unique among the processors hosted by that Network Element.

An initial VG configuration must be specified for the system. It must be compiled into the operating system and loaded into the Network Elements. Both the NE and OS versions of the configuration specify each VG's redundancy level, its Processing Elements, a fault mask, and a communication timeout value. The processor class of the VG, its rate group phasing, and the physical identifier of each of its members is also included in the OS configuration. The processor class field is used to identify the ports associated with each processor class and limit the contention for these ports to processors of the appropriate class during initialization. This prevents different processor types from becoming members of the same VG. The rate group phasing specifies the relative phase of the VG's rate group frames to the frames on other VGs. The physical identifier of each Processing Element is the only field which is not specified in the initial configuration. It is determined after the Processing Elements has successfully contended for a communication port and are a member of the VG mapped to that port.

The rate group phasing describes the relationship between the rate group frames on each VG in the system. Within the task configuration table described in the next section, each task is assigned to execute in some rate group. The rate group determines the frequency at which the task will be executed and the resulting rate group frame delimits the execution cycle of the task. Tasks assigned to the same rate group will execute at the same frequency regardless of their hosting VG, but there may be a time difference between the start of their first and each subsequent rate group frame if the tasks are executing on different VGs. This phasing would be caused by the completion of system initialization at different times on different VGs. An example phasing of the frames for tasks in a given rate group on multiple VGs is shown in Figure 5-3.

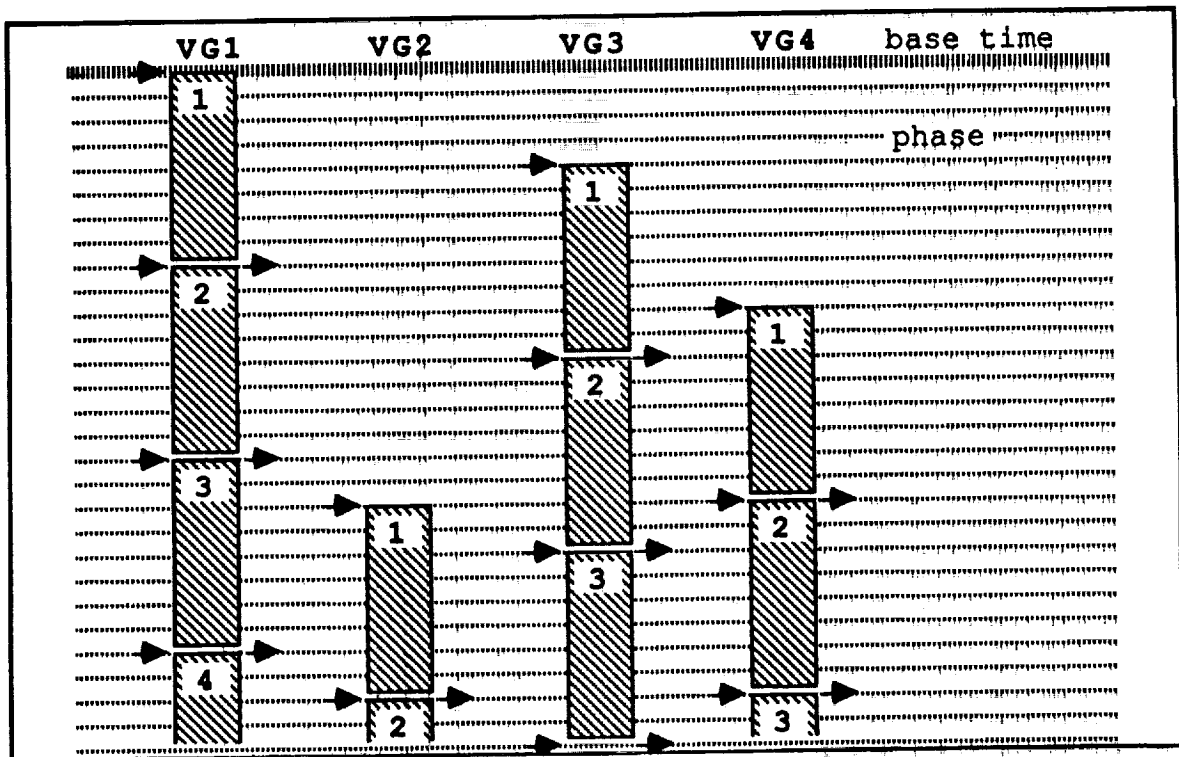


Figure 5-3. Rate Group Frame Phasing

In the example, the first rate group frame on VG1 starts at the base time and the first rate group frames on the the remaining VGs are delayed. The interval between the base time and the start of the first rate group frame is the VG's phase delay. The phase delay is important because it determines the relationship between the frame in which messages are sent and the frame in which they are received. This is also affected by the message passing restrictions in the rate group tasking paradigm. In the paradigm, a task's queued messages are only sent and its received messages are only made available at its corresponding rate group frame boundary. This is indicated in the figure by the arrows at the frame boundaries. An example from the figure is the messages transmitted after the first frame on VG1. They will be received at the start of the first frame on VG2 and VG4, but will not be received until the start of the second frame on VG3. This relationship of sending frame to receiving frame will remain constant for subsequent frames if the phasing does not change.

The phasing will change if the start of subsequent rate group frames on different VGs are allowed to float with respect to each other. The time management service has been designed to minimize this float by locking the phase to the system time maintained by the Network Element. There still remains inherent float because of the variability of the interval from the start of the frame to when any given message will be sent or read. This float is increased when VGs which share a Network Element have the same phase delay or their delays differ by an integer number of minor frames. This is because the VGs are then forced to compete for access to the Network Element to send and read their messages at

their frame boundary. For this reason the simplest phasing of a zero phase delay for all VGs is not recommended. The phase field in the VG configuration table is provided to specify the desired phase delay for each VG. The declaration of the OS version of the configuration table and its associated data types are shown in Figure 5-4.

```

type redundancy_level_type is 0..4;
type processor_class_type is (sbc,intelligent_io);
type minor_frame_period_unit is delta 0.01 range 0.00 .. 8.00;
subtype ne_id_type is (A..E) of message_exchange_type;
type port_id_type is (0..7);
type processor_id_record is record
    ne_id : ne_id_type;
    port_id : port_id_type;
    board_id : integer;
end record;

type vg_configuration_record is record
    redundancy : redundancy_level_type;
    class : processor_class_type;
    phase : minor_frame_period_unit;
    id : array (natural range 1..redundancy_level_type'last) of
        processor_id_record;
    mask : mask_type;
    timeout : timeout_type;
end record;

type vg_id_type is range 0..63;
vg_configuration : array (vg_id_type) of vg_configuration_record;

```

Figure 5-4. VG Configuration Table

vg_id_type defines the allowed set of VG ids and the configuration table has an entry for each VG id. *redundancy* specifies the redundancy of the VG. Zero indicates there are no members in the VG. One, three, and four indicate a simplex, triplex, and quadruplex respectively. Duplexes and quints are not supported. *class* is the processor class of the VG and corresponds to a type of single board computer or intelligent io device. *phase* is phase of the rate group frames on the VG with respect to the frames of other VGs in the system. It is specified in minor frame period units and the maximum phase is one major frame. *id* defines the Network Element and port assignment of each member. It also con-

tains each member's board id for physical identification of the corresponding processor. The board id is determined during initialization based on the results of the port contention process. *mask* and *timeout* specify the fault mask and communication timeout for the VG.

The information in the VG configuration table is sufficient to complete the first phases of system initialization. These phases are processor initialization, Ada elaboration, port contention, FCR initialization, and Network Element synchronization. Processor initialization is performed by each processor at power up to initialize the local hardware devices and perform self tests. The test results are written to an assigned area of mass memory. Processors which deem themselves healthy perform Ada elaboration and begin execution of the *main* task. The remaining processors attempt to remain passive throughout the remainder of the mission. Within main *each* processor determines its processor class and its hosting Network Element. It then evaluates the configuration table to determine the network element ports assigned to processors of its class on this Network Element. All the processors in each class then contend for the assigned ports in ascending order. Based on the acquired port and the hosting Network Element, each process can determine its VG id from the configuration table.

Port 0 on each Network Element is a specially designated port. The processors which acquire these ports are responsible for testing and initializing the shared components of their FCR and any dumb components within their FCR. These test results are also written to mass memory. When the testing is complete, these processors direct the network elements to attempt initial synchronization. A Network Element can also be programmed to spontaneously attempt synchronization. This is useful if the NE does not host any processors. The processors which did not acquire port 0 remain passive waiting for directives from the system manager VG. When synchronization is successful, the Network Elements start system time keeping and are capable of providing inter-VG communication.

5.2.2. Rate Group Task Configuration

The remainder of system initialization requires the use of the task configuration table. The task configuration table maps tasks to VGs and specifies the task's rate group assignment and message buffering requirements. Each task is assigned a communication id or CID and has a corresponding entry in the task configuration table. The CID is used as the task's logical address for intertask communication. An initial configuration must be specified for each task in the system and compiled into the OS. The structure of the task configuration table is shown in Figure 5-5.

```

type location_type is (no_vg, one_vg, all_vg);
type rg_type is (RG1, RG2, RG3, RG4);
type precedence_type is range 0..15;

type task_configuration_record is
  location : location_type;
  vg_id : vg_id_type;
  rg : rg_type;
  precedence : precedence_type;
  task_id : task_id_type;
  max_xmit_size : natural;
  max_xmit_num : natural;
  max_rcve_size : natural;
  max_rcve_num : natural;
end record;

type communication_id_type is ( rg_dispatcher,
                                fdi,
                                system_fdi,
                                ...,
                                appl1,
                                appl2);

task_configuration : array (communication_id_type) of
  task_configuration_record;

```

Figure 5-5. Task Configuration Table

rg_dispatcher, *fdi*, *system_fdi*, *appl1*, and *appl2* are tasks in the system and each task has an associated configuration record. *location* is used to define whether the associated task is not executing, executing on one VG, or executing on all VGs. If the task is not executing, then the remaining fields are invalid. If the task is executing on only one VG, then *vg_id* defines which VG. *rg* defines in which rate group the task is executing. *precedence* is the task's precedence among the tasks executing in the same rate group on the same virtual group. It is used to determine their execution order within the rate group frame. The highest precedence corresponds to 15. *task_id* is the VG's local identifier for the task. Unlike the communication ids, the task ids are not guaranteed to be unique throughout the system. The maximum number and maximum size of messages that the task will have queued for transmission is indicated in *max_xmit_num* and *max_xmit_size*. Their *rcve* counterparts are used for messages that it will have queued waiting to be read.

An initialization must be specified for each entry in the table within the enclosing package body. A partial initialization of the above example is shown in Figure 5-6. In this example the rate group dispatcher has been installed to execute on all VGs as an RG4 task with the highest precedence and an application rate group tasks has been installed to execute on VG 4 as an RG3 task with high precedence. It is important that the rate group dispatcher is the first element in the *communication_id_type*, executes as an RG4 task, and has the highest precedence. This guarantees that it will execute at the beginning of each minor frame and be able to dispatch the other rate group tasks.

```
task_configuration(rg_dispatcher) := (
    location => all_vg,
    rg => RG4,
    precedence := precedence_type'last,
    task_id => rg_dispatcher_id,
    max_xmit_size := 40,
    max_xmit_num := 10,
    max_rcve_size := 40,
    max_rcve_num := 10);
task_configuration(appl1) := (
    location => one_vg,
    vg_id => 4,
    rg => RG3,
    precedence := 12,
    task_id => appl1_id,
    max_xmit_size := 50,
    max_xmit_num := 5,
    max_rcve_size := 10,
    max_rcve_num := 2);
```

Figure 5-6. Task Configuration Table Initialization

The VG which is hosting *system_fdi* is designated the system manager and is responsible for the directing the remaining phases of system initialization. These phases are system manager alignment, resource evaluation and reconfiguration, VG initialization, and system start. System manager alignment is performed by the system manager to align the memory and devices of its members. The system manager then performs resource evaluation by testing the Network Elements and polling each VG in the system for its members' test results and physical identifiers. The physical identifiers are recorded in the VG configuration table and the test results are analyzed to determine which VGs have faulty members.

Reconfiguration is performed if there are unutilized simplex VGs which can be used to replace the faulty members. If there are sufficient resources after reconfiguration to meet the mission requirements then the system manager begins VG initialization.

VG initialization consists of aligning the memory and devices of each VG and initializing their rate group tasking services, communication services, and time management services. The system manager directs each VG to perform the above initialization and waits for confirmation that the initialization was completed. Rate group tasking initialization uses the task configuration table to determine the locally executing tasks and installs them into the local rate group tasking suite. The communication services initialization allocates the packet queues required by the local tasks and enables message based communication. Time management initialization starts rate group tasking with the phase specified in the VG configuration table when the directive to start operational execution is received from the system manager. This directive is a broadcast to all VGs and its timestamp defines the reference time for the rate group phasing. System initialization is completed when each VG receives the directives and begins rate group tasking.

5.3. Rate Group Tasking Services

Within a VG of the AFTA, multiple tasks require the use of the message passing resource. These include both application tasks and timer based preemptive Ada Run Time System (RTS) services. In order to maintain congruent use of this resource across the members of a VG, it is necessary to ensure there is no competition for its use. This is done by limiting the preemption allowed in the system and by limiting the use of the message passing resource. A rate group tasking paradigm was developed to fulfil these requirements for the AFTA. The paradigm consists of the AFTA rate group tasking services, the AFTA time management services, and the AFTA communication services.

Within the rate group tasking services, tasks are assigned to execute as either RG1, RG2, RG3, or RG4 tasks and a rate group dispatcher is provided to control their execution. The tasks in each rate group must be cyclic and execute one complete iteration within their rate group frame. The time management services only allow task preemption at the fastest rate group's frame boundaries. At these boundaries, the rate group dispatcher preempts the executing task and starts the execution of tasks in faster rate groups. The communication services are provided to prevent preemptible tasks from using the message passing resource directly. Instead, their messages are buffered on queues controlled by the rate group dispatcher. This removes the possibility of contention for the message passing resource.

The rate group tasking initialization and associated interaction with the time management and communication service initialization are described in Section 5.3.1. The rate

group dispatcher is described in Section 5.3.2 and the structure of rate group tasks is described in Section 5.3.3. The time management services will be described in Section 5.4 and the communication services will be described in Section 5.5.

5.3.1. Rate Group Tasking Initialization

Every task installed in the task configuration table must be present on each VG and will start execution during elaboration. The tasks must suspend themselves until processor and Network Element initialization are completed and the local VG id has been determined. Based upon the VG id, a list of the tasks executing locally is created and these tasks are scheduled for execution. The tasks not executing locally will not be scheduled and will remain suspended throughout the mission. After the indicated tasks are scheduled, the communication services and time management services are initialized. When the initialization is complete, the rate group dispatcher will begin execution of the first rate group frame and trigger the execution of the appropriate rate group tasks.

The list of tasks executing locally is created from the task configuration table and is maintained as a separate linked list of the tasks in each rate group. The head of each list is stored in the *rg_task_lists* structure. *rg_task_lists* is used during initialization to set up the scheduling parameters for the tasks and to allocate packet buffers for the locally executing tasks. After initialization, it is used at each rate group frame boundary by the rate group dispatcher to check the overrun status of the tasks and by the communication services to transmit their messages and to update their frame markers. The declaration of the rate group task lists is shown in Figure 5-7.

```

type rg_task_record;
type access_rg_task_record is access rg_task_record;
type rg_task_record is record
    task_id : task_id_type;
    cid : communication_id_type;
    next_task : access_rg_task_record;
end record;
type rg_task_list_array is array (rg_type) of access_rg_task_record;
rg_task_lists : rg_task_list_array;

```

Figure 5-7. Rate Group Task Lists

task_id is the local run time system identifier for the task. It is used for scheduling and checking the execution status of the tasks in each list. *cid* is the communication id for the

task and is used to allocate the task's packet buffers and maintain its queues. *next_task* is a pointer to the next entry in the list. *next_task* is *null* in the last list entry of each rate group.

The *init_rate_group_tasking* procedure creates the rate group task lists and sets the scheduling parameters for the rate group tasks. It is called by the *main* task after the local VG id has been determined. The order in which tasks are placed in the lists determines their execution order within the rate group frame. The *precedence* field in the task configuration entry is used to determine this order. Tasks with higher precedence will execute before tasks with lower precedence. If tasks have equal precedence, then the task first declared in the *communication_id_type* will execute first. The *init_rate_group_tasking* procedure declaration is shown in Figure 5-8.

```
procedure init_rate_group_tasking;
```

Figure 5-8. Initialize Rate Group Tasking Procedure

After *init_rate_group_tasking* creates the task lists it then initializes the scheduling parameters for each task in the lists. The rate group dispatcher will be the first task in the RG4 list. Its execution priority is set to *rg_dispatcher_priority* and it is set to start execution when the *start_rg_tasking* event is set and to thereafter resume execution after every minor frame period interval. The execution priorities of the remaining tasks in the RG4 list are set to *rg4_priority* and they are set to resume execution whenever the *rg4_event* is set. The priorities of the tasks in the RG3, RG2, and RG1 lists are set to *rg3_priority*, *rg2_priority*, and *rg1_priority* respectively and they are likewise set to resume execution whenever the *rg3_event*, *rg2_event*, and *rg1_event* is respectively set.

The task priority is used to provide preferred execution of the rate group dispatcher and tasks in faster rate groups when tasks in multiple rate group are executable. The priority declaration is shown in Figure 5-9. Lowest priority is 0 and highest priority is 15.

<i>main_priority</i>	: constant := 10
<i>rg_dispatcher_priority</i>	: constant := 9;
<i>rg4_priority</i>	: constant := 7;
<i>rg3_priority</i>	: constant := 6
<i>rg2_priority</i>	: constant := 5;
<i>rg1_priority</i>	: constant := 4;

Figure 5-9. Task Priority

After the return from *init_rate_group_tasking*, *main* calls *init_communication* and *init_timekeeping*. *init_communication* initializes the packet queues used by the communication services and is described in Section 5.5.2. *init_timekeeping* sets up the local VG's rate group frame phase and returns when the time management service has been started. It is described in Section 5.4.1.

The *start_rg_tasking* event is set by *main* after the return from *init_timekeeping*. All the rate group tasks have suspended themselves during elaboration and are waiting to be resumed based upon their scheduling parameters set up in *init_rate_group_tasking*. When *start_rg_tasking* is set, the rate group dispatcher is placed on the RTS ready queue and will begin execution when the higher priority *main* task completes. At the event, the rate group dispatcher is also placed in the RTS delay queue and will be resumed by the RTS every minor frame period thereafter. After setting the event, *main* is suspended, the rate group dispatcher resumes execution, and rate group tasking has started.

5.3.2. Rate Group Dispatcher

The rate group dispatcher is a special RG4 task that is responsible for controlling the execution of the rate group tasks and providing reliable communication between rate group tasks throughout the system. It executes at the start of each minor frame and based upon the minor frame index determines the corresponding rate group frame boundaries. It checks that the tasks in these rate groups have completed an iteration of their execution cycle and uses the communication services to transmit the messages queued by these tasks and to update the set of messages available for their retrieval. It then sets the events to trigger the next execution cycle of these tasks and suspends itself. These rate group tasks and any slower rate group tasks which have an execution cycle still in progress then resume execution based upon their assigned rate group and precedence within the rate group. The mapping of rate group frames to minor frames is shown in Figure 5-10.

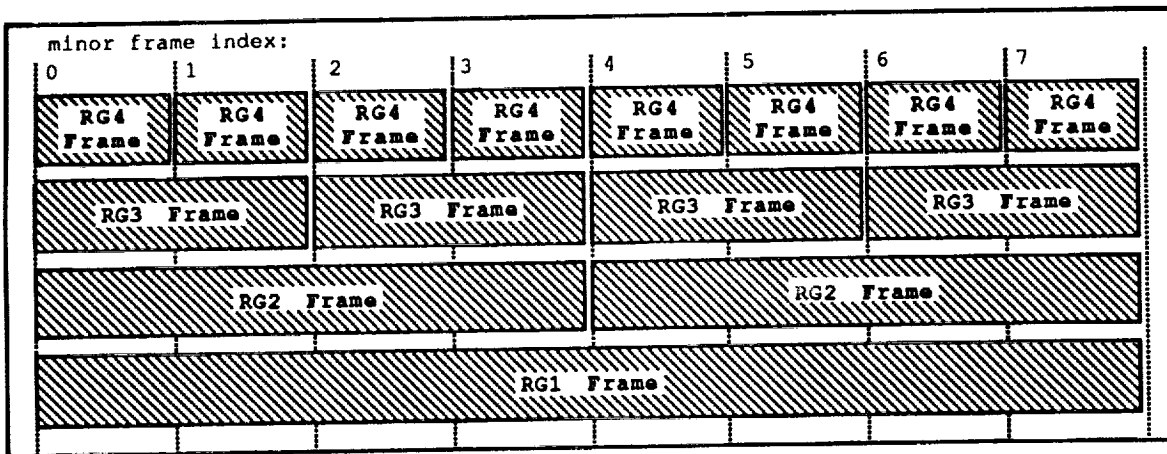


Figure 5-10. Mapping of RG Frames to Minor Frames

During elaboration the rate group dispatcher suspends itself and will not be resumed until the *start_rg_tasking* event is set. When the *start_rg_tasking* event is set and the higher priority *main* task suspends itself, the rate group dispatcher will begin execution of its first cycle and will repeat its execution cycle every minor frame period thereafter. At the start of each cycle, the rate group dispatcher records a congruent value of the current time. It then determines the slowest rate group whose frame boundary corresponds to the start of this minor frame. Because of the mapping of rate group frames to minor frames, all faster rate groups will also be at a frame boundary and the identifier of the slowest rate group is used to indicate the entire set of rate groups at a frame boundary.

The *send_queue* and *update_frame_marker* communication services are then called and passed the identifier of the slowest rate group at the frame boundary. *send_queue* transmits all the messages enqueued by the tasks of the corresponding rate groups in their previous frame. *update_frame_marker* updates the communication service pointers to provide a congruent set of received messages and free buffers to the rate group tasks throughout their frame. *send_queue* and *update_frame_marker* are described in Section 5.4

frame_start is then called and passed the slowest rate group identifier and the time recorded by the rate group dispatcher at the start of this execution cycle. It uses the time value to update the time latch for each of the corresponding rate groups. A time latch is provided for each rate group and is used to latch the time of the start of the rate group frame. It is the only value of time which is guaranteed to be congruent during the task's execution and the only value of time which should be used by rate group tasks. *frame_start* then uses the rate group task lists to determine the tasks executing in the indicated rate groups and checks the overrun condition of each of the tasks. If a task has overrun the condition is logged in the rate group dispatcher log. The log can be examined from the terminal display. It then sets the appropriate *rg4_event*, *rg3_event*, *rg2_event*, or

rg1_event to ready the tasks in the indicated rate groups for their next execution cycle. The *frame_start* procedure declaration is shown in Figure 5-11.

```
procedure frame_start(  
    slowest_rg : in rg_type;  
    congruent_time : in time);
```

Figure 5-11. Frame Start Procedure

After the *frame_start* procedure is executed, the rate group dispatcher then increments the minor frame index and suspends itself until the start of the next minor frame. This allows the lower priority tasks which were previously executing or which were readied for execution by setting the rate group events to begin execution based upon their priority and precedence.

The RG1 frame boundary is a special condition. Because all rate groups have a frame boundary at the RG1 frame boundary, this point defines where the memory used by the rate group tasks will be congruent on all members of the VG and can be successfully aligned. It is only at these points that the rate group manager will attempt to recover a failed processor by invoking *lost_soul*. If *lost_soul* is required, then it will be called by the dispatcher after *update_frame_marker* at the start of the first minor frame. If no channel is recovered in *lost_soul* then the remainder of the frame should proceed normally. If a channel is recovered then some frames may be slipped because of the recovery process. A detailed description of processor recovery is contained in Section 5.6.

5.3.3. Rate Group Tasks

Rate group tasks must be uniquely associated with a communication id and a corresponding task configuration table entry as described in Section 5.2.2. The table entry must be initialized to specify whether the task is executing on one VG or executing on all VGs. System service tasks normally execute on all VGs. If a task executes on all VGs then broadcast messages can be used to send a message to all instantiations of the task. Otherwise the task instantiation must be identified by specifying the hosting VG id. If a task will execute on only one VG then that VG must be specified in the table and the tasks communication id is sufficient to uniquely identify the task. The task's rate group and precedence within the rate group must also be specified. This determines how often the task will execute and the order in which tasks in the same rate group and on the same VG will execute. The local RTS identifier must also be specified to provide the link between the logical communication identifier and the actual task.

The maximum number and maximum size of messages that each task will queue for transmission and that may be queued for its reception must be specified. These values are used to allocate packet buffers for the task's messages. Each task has private and separate outgoing and incoming message queues. A given task's queue operations (including overflows) have no effect on the state of other tasks' queues. If an executing task attempts to enqueue a message to a full outgoing message queue, an error indication is immediately returned to that task, with the outgoing queue and message to be enqueued being left unchanged. In the event of incoming queue overflows, the AFTA operating system indicates the number of incoming messages that have been discarded to the task which would have received the messages had the incoming queue overflow not occurred. Tasks should be designed to check this indication of discarded incoming messages and perform appropriate application-specific recovery from this error condition. An example of such a recovery policy would be to utilize stale input data instead of input data derived from the discarded input message. Note that all members of a redundant VG have an identical view of both outgoing and incoming message queue overflow conditions. In addition, tasks are never presented with a message queue containing partial messages; the AFTA operating system ensures that complete messages are delivered from one task to another in the absence of queue overflows, or no message whatsoever is transmitted.

The task itself must have a well defined cyclic execution behavior. The task and all the other tasks specified to execute on the VG must complete their execution cycle within their rate group frame. If they do not, then the rate group dispatcher will detect an overrun condition for those tasks which did not complete within their frame. These tasks are not necessarily the ones that caused the overrun condition to occur. Rate group tasks may use the *queue_message* and *retrieve_message* to communicate between tasks. Both the reception and transmission of the communication is based on the rate group frame boundary as described in Section 5.5. This must be accounted for in determining the communication timing and the message allocation.

An example of a minimal task is shown in Figure 5-12. Associated with the task declaration is the task id declaration used in the task configuration table initialization. The task defines *my_cid* to be its associated communication id. This is used in the communication and rate group tasking procedures calls to identify the calling task. *num_deleted* is used to indicate how many of the task's messages were deleted in the previous rate group frame because of insufficient free packet buffers. The allocation specified in the task configuration table initialization should be used to ensure that message are not deleted. *frame_time* maintains a congruent value of the time the current rate group frame was started. It should be the only value of current time that is used by the task.


```

task appll_task is
end task;
appll_id : task_id_type := id(appll_task'address);

task body appll_task is
    my_cid : constant communication_id_type := appll;
    num_deleted : natural := 0;
    frame_time : time := startup_time;
begin
    loop
        wait_for_next_frame(my_cid,num_deleted,frame_time);
    end loop;
end task;

```

Figure 5-12. Example Rate Group Task

The task begins execution during elaboration and may perform data initialization. The hosting VG id is not known at this time and the initialization will occur on all VGs even if the task will not be executing on a particular VG. It must suspend itself using *wait_for_next_frame* to end its task elaboration. Based on the local VG id and the task configuration table, the task instantiations are then selectively resumed when rate group tasking begins. When the task is resumed it returns from *wait_for_next_frame* with the *num_deleted* and *frame_time* values updated. The task should then begin its cyclic execution. At the end of its cycle it must again call *wait_for_next_frame* to perform its self suspension. The *wait_for_next_frame* procedure declaration is shown in Figure 5-13.

```

procedure wait_for_next_frame(
    cid : communication_id_type;
    deleted_messages : out natural;
    frame_time : out time);

```

Figure 5-13. Wait for Next Frame Procedure

wait_for_next_frame uses *cid* to identify the rate group of the caller and access its *cid_status_record* maintained by the communication services. When *wait_for_next_frame* is executed it suspends the calling task. To resume execution, the rate group dispatcher must set the appropriate rate group event. Prior to setting the event, the dispatcher can use

the RTS to examine the execution status of the task. If it is not suspended then an overrun condition exists. When the event is set the task resumes execution inside the *wait_for_next_frame* procedure. It uses the congruent time latches maintained by the rate group dispatcher to update *frame_time* and the communication service's *cid_status_record* to update *deleted_messages*. *wait_for_next_frame* then returns to the calling task to begin the next execution cycle.

5.4. Time Management

The time management service is executed on each processor in the AFTA to maintain congruent execution between the members of a VG in the presence of timer based events and to provide a consistent system time for all the VGs. The system time is maintained by the Network Element aggregate as the elapsed time since system start up. When a packet is received by a Network Element the system time is written to the packet's corresponding descriptor field. This time information is used by the time management service to define absolute time. Each processor in the AFTA locally maintains a timer to measure the elapsed time since the absolute time was updated. This timer is used to generate periodic interrupts to define the start of each minor rate group frame and to trigger the update of the absolute time. The interrupt causes preemption of the currently executing task by the rate group dispatcher. The execution state of the system must be well defined at these points to maintain congruent execution. This is provided by the rate group tasking implementation.

The system time maintained by the Network Element and its timestamp of delivered packets has been discussed previously. Section 5.4.1 will discuss the initialization of local time management on each processor of the AFTA. Section 5.4.2 will discuss the timing model used by the RTS and the operation of the time management service.

5.4.1. Time Management Initialization

System time keeping is started by the Network Elements during Network Element initialization and is maintained by the Network Elements throughout system operation. The time management service on each processor of the AFTA is closely coupled with their execution of the rate group tasking paradigm and must not be started until the VG is ready to begin operational execution of the paradigm. Prior to that time each VG is waiting for initialization directives from the system manager VG. These messages do not go through the communication services and are read synchronously by the *main* task. The timestamps associated with these message are used to update the local value of absolute time, but the local timers are not active and no timer based interrupts are generated.

The VGs will receive the system manager directive to begin operational execution after all other VG initialization has been completed. This message will be a broadcast to all VGs and the timestamp of the message will be used as the base time for starting rate group tasking

on all VGs with the phasing specified in the rate group phase table described in Section 5.3.1. When the message is received, each member of a VG sets its local timer to generate an interrupt based on the VG's assigned phase. When the interrupt is generated, the first frame of rate group tasking is started and subsequently an interrupt will be generated every minor frame period. The time management service now begins operation and is responsible for coordinating the local timer with the Network Element system time and maintaining the phasing specified in the phase table. Its operation is described in the next section. *init_timekeeping* is the procedure responsible for starting the time management service with the appropriate phase. Its declaration is shown in Figure 5-14.

```
procedure init_timekeeping(phase : in time);
```

Figure 5-14. Initialize Time Keeping Procedure

init_timekeeping is called from the *main* task after all other VG initialization has been completed and the directive from the system manager to begin operational execution has been received. The timestamp of this message is the reference time for the phasing of rate group frames across all VGs. The reading of the message has also updated the local value of absolute time to this value. *init_timekeeping* is called in response to the directive and is passed the delay corresponding to the phase specified for the VG in the rate group phase table. It sets the local timer to generate an interrupt when the delay has expired and enables generation of the receive message interrupt from the Network Element. Messages can now be read asynchronously and will be processed by the communication services. *init_timekeeping* then suspends itself until after the interrupt. When the interrupt is generated the time management procedures described in the next section begin execution.

Normally the rate group dispatcher would be resumed immediately after the interrupt. At the first timer interrupt only, the *main* task is still executable and will be resumed after the interrupt. *init_timekeeping* will return to *main* and it will set the *start_rg_tasking* event to start execution of the rate group dispatcher. *main* will then terminate its execution and allow the lower priority rate group dispatcher to execute. It is necessary to coordinate the start of time management and the execution of the rate group dispatcher to ensure that the dispatcher and the rate group tasks have a full minor frame period to execute. Otherwise, our task completion guarantees are not valid and a frame overrun condition may result.

5.4.2. Time Management Operation

After the time management service has been started, a chiming model is used to maintain the local value of system time on each processor of the AFTA. The local timer is used

to generate the chime every minor frame period on each member of a VG. When the chime is generated, the VG members resynchronize and congruently update the local value of system time with the system time maintained by the Network Element. The updated time may not agree with the expected time of the chime and this difference is used to adjust the next chime interval and maintain a constant frame phasing among the VGs in the system. The local value of system time is only updated at the chime interrupt and is the only time value provided to the remainder of the RTS.

Time management is provided by the chime interrupt handler. It will execute at each minor frame boundary and the rate group tasking paradigm must guarantee that no packet transmissions are in progress on any of the VG members when the chime is generated. Packet receptions may be in progress because of the asynchronous packet reception provided by the communication service. The packet reception interrupt has higher priority than the chime interrupt and will execute to completion prior to the chime interrupt being serviced. When the chime interrupt handler is executed, it disables the packet reception interrupt and sends itself a synchronization packet to flush all received packets from its Network Element buffers. It reads and handles packets in the same manner as the packet reception handler until the synchronization packet is found. It then sends itself an additional synchronization packet through the flushed network and uses the timestamp of this packet to update the local copy of system time.

The chime interrupt handler then determines the expected time of the next chime interrupt by adding the minor frame period interval to the expected time of the current chime. It records this value and sets the next chime to generated after an interval corresponding to the difference between this value and the system time last read from the Network Element. This will maintain the rate group phase relationship between the VGs. The chime interrupt handler then enables the Network Element packet reception interrupt and returns to the RTS. The RTS reevaluates the scheduable tasks based on the updated time. The rate group dispatcher will now be placed on the ready queue and resume operation because it is the highest priority ready task.

5.5. Communication Services

The communication services are used to communicate between rate group tasks. Each rate group task has a global communication id which can be used as its logical address. Other tasks in the system can send messages to this address and the communication services will map the logical address to the VG executing the task. The communication is in the form of messages enqueued by the sender for transmission at the start of the next rate group frame and dequeued for reading by the recipient task within the next rate group frame after it is received. Messages are delivered in the same order at all common destinations and are delivered in the order in which they were sent.

The communication services are composed of message based interface procedures used by rate group tasks and lower level primitives used by the rate group dispatcher. The communication service primitives manipulate the messages as a sets of queued packets. The message and corresponding packet structures are described in Section 5.5.1. The communication service initialization and associated control structures are described in Section 5.5.2. The message transmission procedures are described in Section 5.5.3 and the message reception procedures are described in Section 5.5.4.

5.5.1. Message and Packet Structure

The rate group tasks have a message based interface to the communication services. The message itself is a contiguous block of data that is transferred from the sender to the receiver. The block must be no larger than the maximum message size defined for the system. Associated with the message are descriptor fields describing the sender, receiver, type of message, and how the message is to be exchanged. The message and message descriptor fields are supplied to the communication services by the task wishing to send a message. The communication services then perform the exchange and deliver the message and descriptor information to the receiving task when it requests delivery of its messages.

Internally, the communication services store and manipulate the message as a set of fixed size packets. A packet is the exchange unit used by the Network Elements. The message descriptor fields are mapped to packet descriptor fields and a message header. The packet descriptors are sent with each packet and the message header is prepended to the message data and sent only in the first packet of the message. The message and packet structure for task-to-task communication is shown in Figure 5-15.

The message descriptors consist of the destination VG id, the destination communication id, the source VG id, the source communication id, the message class, and the size of the message data. The destination VG id is supplied by the sending task and used in the packet exchange, but is not delivered to the receiving task. The source VG id is not supplied by the sending task, but is provided by the Network Element with each delivered packet. It is provided as a message descriptor to the receiving task.

The VG id is used to specify a virtual group and the communication id is used to specify a task executing on the VG. The message size specifies the size of the message data in bytes. The message class is used to specify whether the message should be broadcast to all VGs, whether the message is task-to-task data or task-to-ne data, and whether the message should be a voted or a single source exchange.

Broadcasts messages are only useful when the destination task has an instantiation on all VGs. In addition, they monopolize bandwidth and can cause flow control problems. For this reason it is the intent to limit the use of broadcast messages to system service

tasks. Task-to-ne data is send by system services to update the NE configuration table, generate voted resets, and perform initial synchronization. All other tasks must send only task-to-task data.

The packet descriptors consist of the destination VG id, the destination communication id, the source VG id, the message class, and a boolean indicating whether this is the last packet of the message. All except the last packet boolean and the source VG id are copied directly from the message descriptor. The packet descriptors are included with every packet and are always voted by the Network Elements. This is true even for single source exchanges where the corresponding packet data is not voted, but congruently replicated from the single source. This guarantees that a single member of a redundant VG cannot cause this information to be corrupted on the delivered packet.

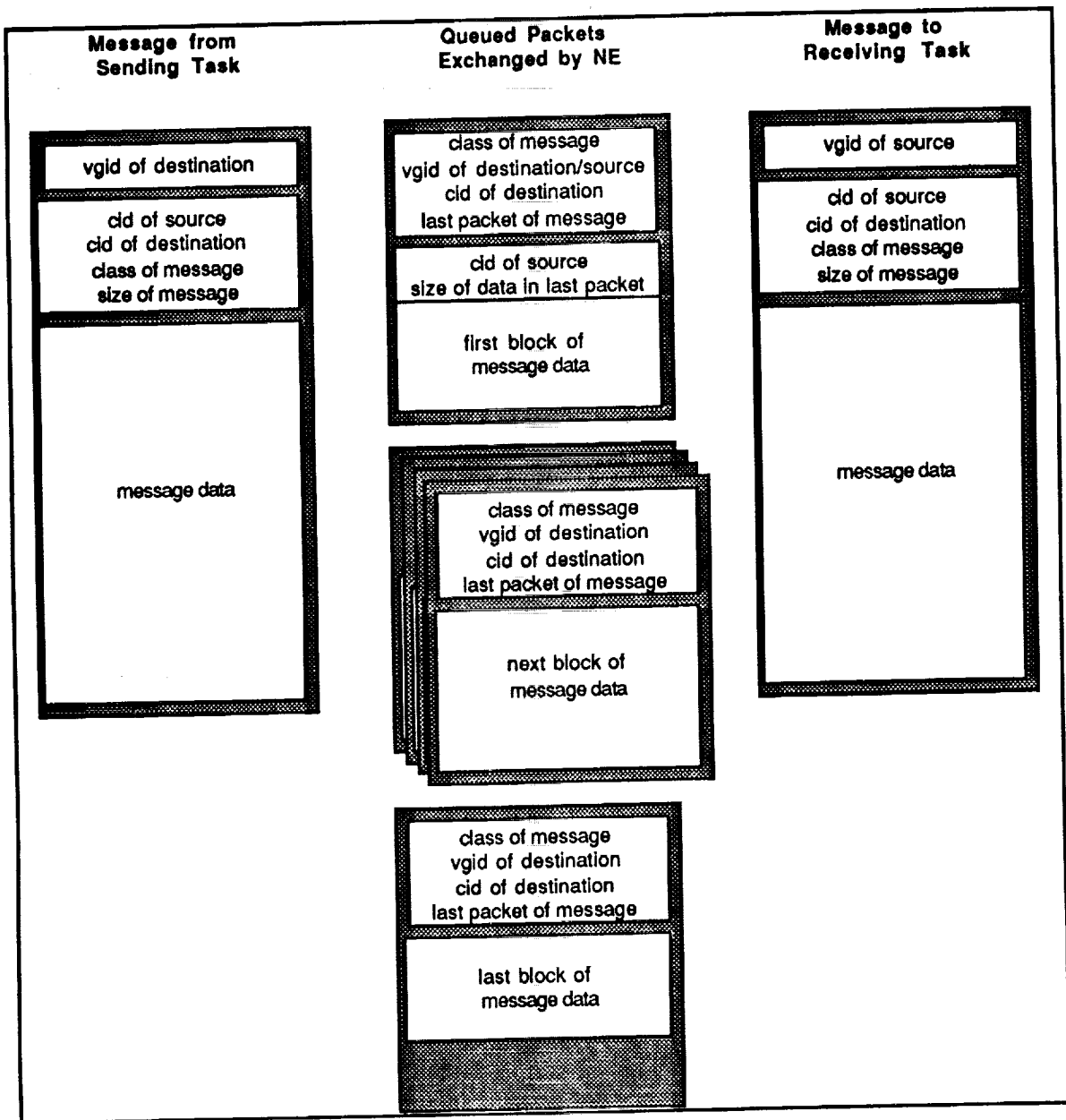


Figure 5-15. Task-to-Task Message and Packet Formats

The source VG id and the last packet boolean are used by the receiving VG's communication services to link the packets of a message together. Within the AFTA, the communication services of each VG will send all the packets of a message before it starts sending the next message. Therefore, all the packets from a VG will belong to the same message until the last packet boolean is true. Then the next packet will be the start of a new message. The destination cid is used by the receiving communication services to determine the correct queue for the packet. The source VG id is supplied by the NE with each delivered packet and is copied to the receive message descriptor.

In addition to the packet descriptors described above, a packet syndrome and packet timestamp are also provided by the NE for each delivered packet. These fields are maintained in the queue of received packets and are used by system services, but they are not propagated to the receiving task.

The message header is only prepended to the message data for task-to-task data and is sent in the first packet of the message. It is not included when task-to-ne data is to be sent. The message header consists of the source communication id and the size of the data in the last packet. The source communication id is used only as information to the receiving task. The size of the data in the last packet is used with the number of packets in the message to determine the message size. This is copied to the receive message descriptor.

The header information is included in the packet data and will not be voted during single source exchanges. A single member of a VG may therefore cause this information to be faulty in the delivered message. Corrupted data will not cause loss of system service, but it may confuse the receiving task. Tasks should be written to tolerate this condition if they expect to receive single source messages.

5.5.2. Communication Services Initialization

A transmit packet queue and a receive packet queue are maintained for each cid. They are the buffers between the underlying packet based communication primitives which directly access the Network Elements and the message based communication services which are used by the rate group tasks. The transmit queues are used to guarantee that the packets written to the NEs by the members of a VG have a consistent ordering. The receive queues are used to guarantee that rate group tasks see a consistent set of available messages. Both these conditions are necessary to guarantee that the members of a VG do not diverge.

Each queue is portioned into a set of active packets followed by a set of free packets. The active transmit packets contain data waiting to be written to the NE. The active receive packets contain data waiting to be read by a task. During initialization all the packets allocated for a task are placed in the free portion of the respective transmit or receive queue. The allocation is based on the values specified for the corresponding task in the task communication table described in Section 5.2. The queues are maintained as linked lists with pointers to the entry at the head of the active portion, to the entry at the head of the free portion, and to the entry at the tail of the free portion.

In the transmit queues, entries are moved from the free portion to the active portion when a message is enqueued by a task. This transition is performed by using the entries at the free head to store the packetized message and then making the free head point to the next free entry in the queue. Entries are removed from the active portion and replaced in the free portion when the stored packets are written to the Network Element. Only the en-

try at the active head is written. After it is written, it is removed from the active head and replaced at the free tail and the active head and free tail are updated. The transmit queues are maintained as singly linked lists. An example of the transmit and receive queues is shown in Figure 5-16.

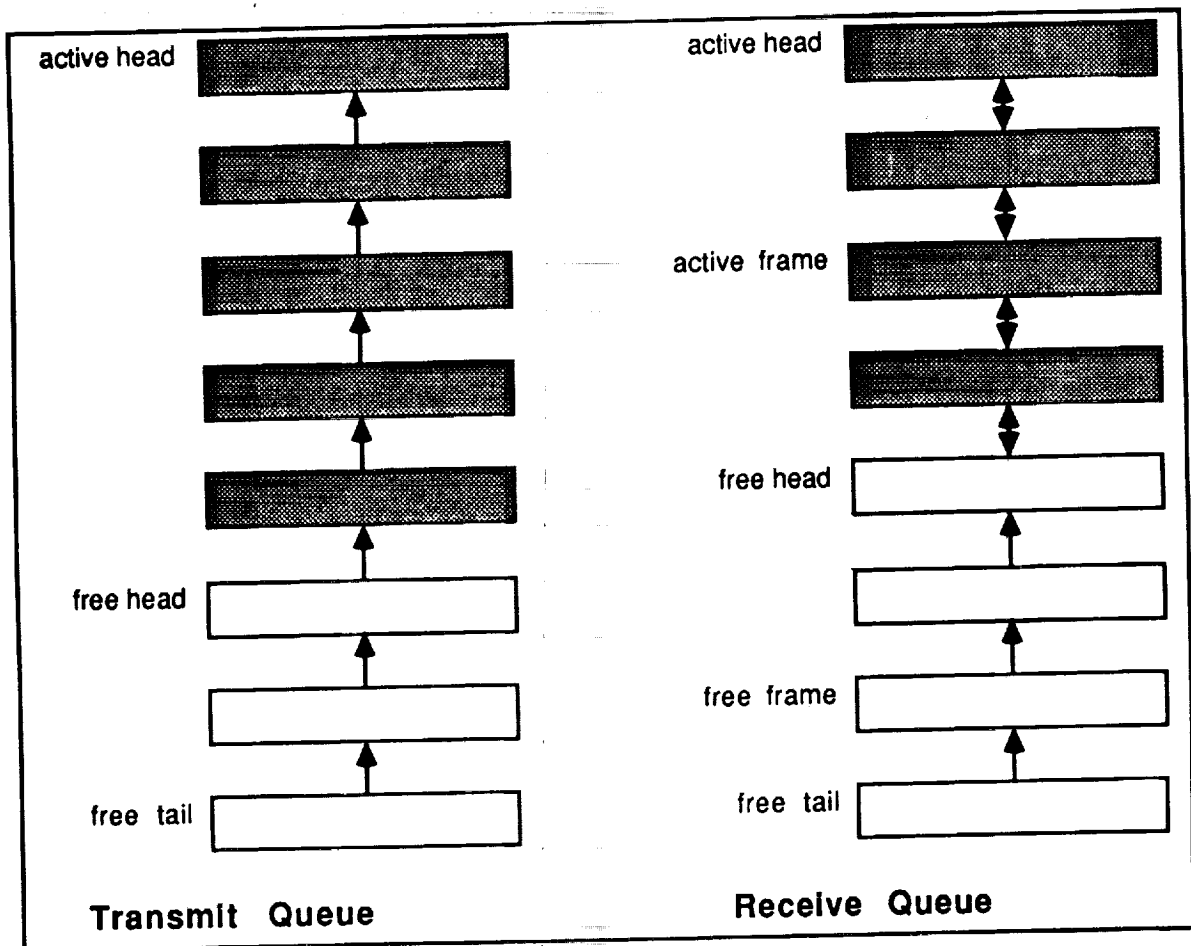


Figure 5-16. Transmit and Receive Queues

In the receive queues, entries are moved from the free portion to the active portion when a packet is read from the Network Element and the transition is the same as in the transmit queues. Entries are removed from the active portion and replaced in the free portion when a task retrieves a message. Because the packets of messages may be interleaved, entries may be removed from anywhere within the active portion. The entries in this portion of the queue are doubly linked so the linked list can be maintained when entries other than those at the active head must be removed. The removed entries are replaced at the free tail and the active head and free tail are updated as necessary.

Maintaining separate queues for each cid guarantees congruent ordering of input and output messages for the corresponding task, but it does not guarantee the timing of the packet events within the rate group frame with respect to other members of the VG. The

only guarantee is that at the rate group frame boundary an identical set of events will have occurred on all members of that rate group. Controlled access to these queues within the rate group frame is necessary to prevent divergence of the VG members. Access to the transmit queues is controlled within the communication service primitives by only writing packets to the Network Element at the rate group frame boundary and then writing all enqueued packets. This guarantees that all the queue entries will be in the free portion of the queue at the start of the frame. The completion of the task within its rate group frame guarantees that each member of a VG will have a congruent set of packets in the active portion of its transmit queue when the packets are sent at the end of a frame.

Unlike the transmit queues, the receive queues may be updated by the communication primitives throughout a rate group frame. This is done whenever the packet reception interrupt is generated by the Network Element. In order to maintain congruent operation, a frame marker is provided for the active portion of each receive queue to indicate the packets which were read from the NE prior to the start of the current frame. The packets between the active head and the active frame marker are guaranteed to be present on all members of the VG and are made available for reading by the task if they compose a complete message. The free portion of each receive queue must also have a frame marker. This is necessary to ensure that a consistent set of free entries is available within the rate group frame for new packets read during the frame's execution. Otherwise, some members of a VG may have no free entries for a packet to a given task, while others do have a free entry and continue normal execution. The entries between the free head and the free frame marker are guaranteed to be free on all members of the VG and are usable to store received packets. The frame markers for a given queue are only updated at the rate group frame boundary for the corresponding task.

```

type xmit_pkt_record;
type access_xmit_pkt_record is access xmit_pkt_record;
type xmit_pkt_record is record
    next_in_queue : access_xmit_pkt_record;
    message_class : message_class_type;
    to_vg : vg_id_type;
    to_cid : communication_id_type;
    last_packet : boolean;
    packet : message_data_record;
end record;

type rcve_pkt_record;
type access_rcve_pkt_record is access rcve_pkt_record;
type rcve_pkt_record is record
    next_in_queue : access_rcve_packet_record;
    previous_in_queue : access_rcve_packet_record;
    first_in_message : access_rcve_packet_record;
    next_in_message : access_rcve_packet_record;
    message_class : message_class_type;
    from_vg : vg_id_type;
    to_cid : communication_id_type;
    last_packet : boolean;
    syndrome : syndrome_type;
    timestamp : timestamp_type;
    packet : message_data_record;
end record;

```

Figure 5-17. Transmit and Receive Packet Queue Entries

The transmit and receive queue entry declaration is shown in Figure 5-17. The transmit and receive queues use *next_in_queue* as their forward link to the next entry in the list and if the entry is in the free portion this is the entry's only valid field. Entries in the active portion of the receive queue use *previous_in_queue* as their backward link to remove entries from the middle of the queue and use *first_in_message* and *next_in_message* to reconstruct packetized messages. *first_in_message* is null except in the last packet of the corresponding message. There it is set to point to the first packet of the message. This information is used to search the queue for the first completed message. *next_in_message* is used to reconstruct the message once a completed message is found. The *previous_in_queue*, *first_in_message*, and *next_in_message* fields are not necessary for the

transmit queues because the messages in the transmit queues are stored in contiguous packets.

The remaining fields are part of the packet descriptor and the packet data as described in the previous section. The message class and message data declarations are shown in Figure 5-18. *message_data_type* indicates whether the data is being sent task-to-task(*task_data*) or task-to-ne(*ct_update*, *voted_reset*, or *init_sync*). *message_exchange_type* indicates whether the data is to have a voted exchange or a single source exchange. *message_data_record* is a discriminant record that can contain the message header for the first packet in a task-to-task message or contain only message data for subsequent packets in the message or for task-to-ne messages.

```

type message_data_type is (task_data,ct_update,voted_reset,init_sync);
type message_exchange_type is (sync,vote,A,B,C,D,E);
type message_class_type is record
    broadcast : boolean;
    data_type : message_data_type;
    exchange_type : message_exchange_type;
end record;

type message_data_record(header: boolean) is record
    case header is
        when TRUE =>
            from_cid : communication_id_type;
            last_packet_size : natural;
            data : array (3..packet_size) of unsigned_byte;
        when FALSE =>
            data : array (1.. packet_size) of unsigned_byte;
    end case;
end record;

```

Figure 5-18. Message Class and Message Data Structure

The pointers used to access each queue are maintained in a queue table. The table is referenced by *cid* and is initialized by the *init_communication* procedure. The table declaration is shown in Figure 5-19.

```

type cid_queue_record is record
  xmit_active_head : access_xmit_pkt_record;
  xmit_free_head : access_xmit_pkt_record;
  xmit_free_tail : access_xmit_pkt_record;
  rcve_active_head : access_rcve_pkt_record;
  rcve_active_frame : access_rcve_pkt_record;
  rcve_free_head : access_rcve_pkt_record;
  rcve_free_frame : access_rcve_pkt_record;
  rcve_free_tail : access_rcve_pkt_record;
end record;
type cid_queue_array is array (communication_id_type) of
  cid_queue_record;
cid_queue_table : cid_queue_array;

```

Figure 5-19. CID Queue Table

init_communication uses the task configuration table and rate group task lists described in Section 5.3 to allocate packet entries for the tasks which will be executed locally. For each of these tasks, entries are allocated from memory based upon their transmit and receive memory requirements specified in the task configuration table. These entries are placed in their respective transmit and receive queues and the remaining queue pointers are initialized. A warning will be generated if there is insufficient memory to allocate the required number of packets. *init_rate_group_tasking* must have been called previously to initialize the rate group task lists. After *init_communication* is called, the communication services are ready to begin operation, but the packet reception interrupt from the Network Element has not yet been enabled. The interrupt is enabled when the directive to begin rate group tasking is received from the system manager VG and *init_timekeeping* is called. When the interrupt is disabled the communication services can be bypassed and this is the operational mode during system initialization. The *init_communication* declaration is shown below.

```

procedure init_communication;

```

Figure 5-20. Initialize Communication Procedure

5.5.3. Message Transmission

Procedures are provided to immediately transmit a message or to enqueue the message for transmission at the end of a rate group frame. Immediate transmission may only be performed by the rate group dispatcher or RG4 tasks and is done using *send_message*. Enqueued message transmission must be done by RG3, RG2, and RG1 tasks. It may also be done by RG4 tasks and the rate group dispatcher. The message is enqueued using *queue_message* and is transmitted by the rate group dispatcher using *send_queue*.

send_message bypasses the transmit queue and directly accesses the Network Element to transmit a message. It must not be preempted, otherwise members of the VG may write different data to the NE and diverge. For this reason only the rate group dispatcher and RG4 tasks are allowed to use *send_message*. These tasks are guaranteed to complete their iteration every minor frame and will therefore not have pending calls of *send_message* when the frame expires. *send_message* should be used only if it is absolutely necessary and under well defined operating conditions. It is especially dangerous if hardware flow control gets asserted because the message transmission (and hence the transmitting task) will be stalled in a busy wait until the flow control condition is cleared. Stalling a non-preemptive RG4 task for an excessive amount of time could result in a cascade of frame overruns. The *send_message* declaration is shown in Figure 5-21.

```
type send_error_flag_type is (no_errors, illegal_message,  
    illegal_destination, inactive_destination);  
procedure send_message is (  
    source_cid : in communication_id_type;  
    destination_cid : in communication_id_type;  
    destination_vg : in vg_id_type;  
    message_class : in message_class_type;  
    message_address : in address;  
    message_size : in natural;  
    error_flag : out send_error_flag_type);
```

Figure 5-21. Send Message Procedure

source_cid is the communication id of the caller. It is included for possible use by the receiving task. *destination_cid* is the communication id of recipient. It is used to access the task configuration table and determine where the destination task is executing. If the task is executing on all VGs then *destination_vg* is used to determine which instantiation of the task should be sent the message. *message_class* is used to determine the type of message

which is to be sent. *message_address* is the starting address of the data to be sent. *message_size* is the size of the data in bytes. The procedure returns the completion status in *error_flag*. *no_errors* indicates the operation was performed. *illegal_message* indicates the data size was not acceptable or the addresses could not be accessed. *illegal_destination* indicates an illegal combination of destination cid, destination VG, and/or message class. *inactive_destination* indicates the destination cid was not active.

queue_message is used by rate group tasks to queue messages for transmission by the rate group dispatcher at the end of their frame. When it is called by a task the source cid is examined to determine where to queue the message. The message size is then examined to determine if there are enough free transmit packet buffers to enqueue the message. If there are, then the packet descriptors and a message header are constructed and the message is parsed and written into the free packet buffers. These packets are then removed from the free list and placed on the task's transmit active queue. If there are insufficient buffers then a failure condition is returned. The *queue_message* declaration is shown in Figure 5-22.

```

type queue_error_flag_type is (no_errors, illegal_message,
    illegal_destination, inactive_destination, insufficient_free_entries);
procedure queue_message is (
    source_cid : in communication_id_type;
    destination_cid : in communication_id_type;
    destination_vg : in vg_id_type;
    message_class : in message_class_type;
    message_address : in address;
    message_size : in natural;
    error_flag : out queue_error_flag_type);

```

Figure 5-22. Queue Message Procedure

The parameters of the *queue_message* are the same as *send_message*. *source_cid* is now also used to determine in which queue the message belongs. An *insufficient_free_entries* flag is provided to indicate there are not enough free packet buffers to enqueue the message.

At the end of each frame, the rate group dispatcher determines the corresponding rate group frame boundaries as described in Section 5.3.2. The dispatcher calls *send_queue* with the slowest rate group at a frame boundary as its parameter. Because of the mapping of rate group frames to minor frames, all faster rate groups are also at their frame boundary. *send_queue* examines the *cid_queue_record* for all the tasks in the indicated rate

groups and transmits all their enqueued messages. Any incomplete messages in the queue are flushed. This indicates a task overrun or other error condition and is logged. The *send_queue* declaration is shown in Figure 5-23.

procedure send_queue is (slowest_rg : in rg_type);

Figure 5-23. Send Queue Procedure

5.5.4. Message Reception

Messages may be read by the receiving tasks using *read_message* or *retrieve_message*. The rate group dispatcher and RG4 tasks may use *read_message*. It is a blocking call that returns the next message for the caller. The message may already be in the receive queue or it may require waiting for additional packets to read from the NE. RG3, RG2, and RG1 tasks must use *retrieve_message*. It returns the next message for the task if it exists in the receive active queue between its head and the frame marker. Otherwise, it returns a failure status. *retrieve_message* is also useable by RG4 tasks and the rate group dispatcher. *update_frame_marker* is a specialized procedure provided to the rate group dispatcher to update the receive queue frame markers for the tasks in the indicated rate groups.

Packets are asynchronously read from the Network Element throughout the VGs execution in response to an NE generated packet ready interrupt. As each packet is received, the communication services read the associated *from_vg* descriptor and examine the associated message pending table entry. The message pending table indicates whether there is a message in progress from the sending VG. If there is, then this packet must belong to that message and it is linked to the previous packets of the message. Otherwise it is the first packet of a new message and the *to_cid* descriptor is examined to place it on the appropriate queue. The message pending table declaration is shown below.


```

type message_pending_record is record
  first_packet : access_packet_record;
  previous_packet : access_packet_record;
  message_deleted : boolean;
end record;
message_pending_table : array (vg_id_type) of
  message_pending_record;

```

Figure 5-24. Message Pending Table

If a message is in progress from the sending VG then *first_packet* points to the first packet of the message. If the *last_packet* descriptor field indicates this is the last packet of the message, then the *first_in_message* field for this packet is set to *first_packet*. This is used to search the queue for completed messages. *previous_packet* points to the previous packet of the message and is used to link this packet with the previous packet of the message. *message_deleted* is a boolean indicating whether the pending message was flushed because of insufficient free packet buffers. If *message_deleted* is true then all subsequent packets of the message are also flushed.

If there are no available buffers to store a received packet then that packet and all the other packets of the message are flushed. The available buffers are those in the recipient task's receive free queue between its head and frame marker. Limiting the task's message storage to its allocated set of buffers controls propagation of the task's flow control problem. Limiting the use of free entries to those between the head and frame marker guarantee congruent behavior between the members of the VG.

When a packet must be flushed, *first_packet* in the message pending table is used to determine if previous packets of the message have been read. If they have, then the associated queue entries are removed from the active portion of the queue and placed at the head of the free portion. This violates the queue paradigm described previously, but minimizes the number of deleted message by allowing reuse of the buffers during the current frame. Otherwise the buffers would be placed at the tail of the free queue and would not be useable until the next frame update.

A counter of the messages deleted in the current frame is then incremented. If this is not the last packet of the message, then *message_deleted* is set to true. All subsequently read packets of the message are deleted and when the last packet of the message is read the message pending table is readied for the start of the next message. The messages deleted counter is maintained in the cid status table shown in Figure 5-25.

```

type cid_status_record is record
    deleted_in_last : natural;
    deleted_in_current : natural;
end record;
type cid_status_array is array (communication_id_type) of
    cid_status_record;
cid_status_table : cid_status_array;

```

Figure 5-25. CID Status Table

The information made available to the tasks about the number of messages deleted must also be based on their rate group frame to guarantee congruent operation. When a message is deleted the *deleted_in_current* field is incremented. At the next frame boundary, this information is transferred to *deleted_in_last* and made available to the task whose messages were deleted.

The frame markers make the asynchronous reception transparent to the rate group tasks except for loss of execution time in the frame and a possible increase in execution skew. The interrupt is disabled when the synchronization packet used by *update_frame_marker* is read. This is necessary to guarantee a consistent state when *update_frame_marker* updates the control structures. After the control structures are updated the interrupt is enabled and the corresponding interrupt handler is again used to read packets during the remainder of the frame.

read_message returns the next available message to the calling task. It must be called only by the rate group dispatcher or RG4 tasks. It first looks for a completed message to the calling task starting from the head of its receive active queue. If a completed message is not found between the queue head and the queue frame marker, then *read_message* may use packets beyond the frame marker. If a completed message is not found in the queue then it waits for packets to be added to the queue by the asynchronous receive packet interrupt handler until a completed message is found. If the last packet of the completed message is past the frame marker, then the frame marker for that queue is updated to the last packet. This is allowed because all the members of the VG are guaranteed to read the same set of packets prior to reading the last packet. Because of the asynchronous packet reception, no conclusion can be determined about packets after the last packet of the message or packets in other queues. The asynchronous reading of packets also disallows updating the task's receive queue frame marker. *read_message* may take an indefinite amount of time and should only be used if absolutely necessary and under well defined operating conditions. The declaration of the *read_message* procedure is shown in Figure 5-26.

```

type read_error_flag is (no_errors,buffer_too_small);
procedure read_message is (
    source_cid : out communication_id_type;
    source_vg : out vg_id_type;
    destination_cid : communication_id_type;
    message_class : out message_class_type;
    message_address : address;
    message_size : in out natural
    error_flag : out read_error_flag_type);

```

Figure 5-26. Read Message Procedure

read_message is passed the *destination_cid*, *message_address*, and *message_size*. *destination_cid* specifies which receive queue to examine. *message_address* and *message_size* describe the buffer in which the task wants its message to be copied. If the buffer is not large enough for the message then *buffer_too_small* is returned in *error_flag* and the actual message size is returned in *message_size*. *source_cid*, *source_vg*, *message_class*, and *message_size* are copied from the received message descriptors.

update_frame_marker is called by the rate group dispatcher at a rate group frame boundary. It is used to congruently update the set of packets useable by the tasks in that rate group within their next frame. When it is called it sets the receive free queue frame marker to the receive free queue tail for all the tasks in the rate group. Because the tasks in this rate group have completed their execution at the frame boundary this provides the same set of free buffers for use when reading packets within the frame. It then sends itself a class0 synchronization packet and waits until this packet is written to its receive queue. When the synchronization packet is read by the receive interrupt handler, the handler disables the packet reception interrupt. This ensures that the same set of packets will have been read on all members of the VG when *update_frame_marker* resumes execution and sees the synchronization packet in its receive queue. *update_frame_marker* then sets the receive active queue frame marker to the receive active queue tail for all the tasks in the rate group and the messages deleted information in the *cid_status_record* for the corresponding tasks is also updated. The *deleted_in_current* field of the *cid_status_record* for the corresponding tasks is copied to *deleted_in_last* and *deleted_in_current* is reset to zero. *deleted_in_last* is then propagated to each task as a return value from *wait_for_next_frame* when it resumes execution. When the data structures have been congruently updated *update_frame_marker* re-enables the packet receive interrupt. The *update_frame_marker* procedure declaration is shown in Figure 5-27.

```
procedure update_frame_marker is (slowest_rg : rg_type);
```

Figure 5-27. Update Frame Marker Procedure

slowest_rg is the rate group frame boundary corresponding to this call of *update_frame_marker*. Because of the dispatching cycle used by the rate group dispatcher, the frame boundary for any given rate group will also be a boundary for all faster rate groups. This property is used by the dispatcher and *update_frame_marker* to remove the need for multiple calls of *update_frame_marker* at a given boundary. Instead the dispatcher calls *update_frame_marker* with an *rg* of the slowest rate group at the boundary and *update_frame_marker* uses one synchronization packet to update the frame markers of the tasks in that rate group and all faster rate groups.

retrieve_message returns the next available message to the calling task which has been read prior to the last frame marker. It can be called by the rate group dispatcher or by any of the rate group tasks. *retrieve_message* looks for a completed message to the indicated *cid* from the head of its receive active queue to its frame marker. If the message is found it is unpacketized and reconstructed at the message address specified in the *retrieve_message* call. The message descriptor fields are then updated and the freed buffers are placed at the tail of the receive free queue. Otherwise, an error condition is returned. The declaration of the *retrieve_message* procedure is shown in Figure 5-28.

```
type retrieve_error_flag is (no_errors,buffer_too_small,no_message);  
procedure read_message is (  
    source_cid : out communication_id_type;  
    source_vg : out vg_id_type;  
    destination_cid : in communication_id_type;  
    message_class : out message_class_type;  
    message_address : in address;  
    message_size : in out natural  
    error_flag : out retrieve_error_flag_type);
```

Figure 5-28. Retrieve Message Procedure

The *retrieve_message* parameters are the same as *read_message* except for the addition of a *no_message* error flag. This is used to indicate no completed message was found in the queue.

5.6. Fault Detection, Identification and Recovery

The AFTA uses hardware redundancy with fault detection and masking capabilities to provide fault tolerance. This inherent fault detection capability is supplemented with traditional self test methods to increase AFTA's coverage of faults.

The fault tolerance provided by the hardware is enhanced by the Fault Detection, Identification and Recovery (FDIR) functions which are part of the AFTA operating system. While the hardware alone in the AFTA could sustain one fault, the FDIR software allows it to sustain multiple successive faults by identifying a faulty component and masking it from system operations. Consequently, the primary purpose of FDIR is to maintain correct operation in the presence of hardware faults. To achieve this, FDIR has four main functions:

- testing of AFTA components, i.e., initiating various test procedures in order to uncover hardware failures.
- identifying a failed component, i.e., detecting a fault, isolating it to a single component and disabling the faulty component.
- performing a remedial operation, i.e., initiating a recovery operation commensurate with system requirements.
- performing transient fault analysis, i.e., determining whether the error was due to a transient fault.

5.6.1. System and Test Modes

Each of the 4 primary functions of FDIR has various alternatives which arise because the system operating conditions vary. Since the FDIR functions must be commensurate with these conditions, numerous options are posed to match these requirements. Consequently, much of the subsequent discussion on FDIR functions will occur within the framework of system modes and test modes.

FDIR functions occur at all stages of the AFTA's operations. As the computing system proceeds through the various system modes from an initial power-on state through a standby mode to a fully operational mode, the testing methodology also evolves through various modes of testing commensurate with the operational constraints. During each testing mode, suites of tests are activated to exercise the AFTA components both individually and systematically as comprehensively as possible. Specifically, there are three test modes (initial built-in test (I-BIT), maintenance built-in test (M-BIT), and continuous built-in test (C-BIT)) and three system modes (power-on, standby, operational or mission critical). Figure 5-29 depicts the interaction of the system modes and the test modes.

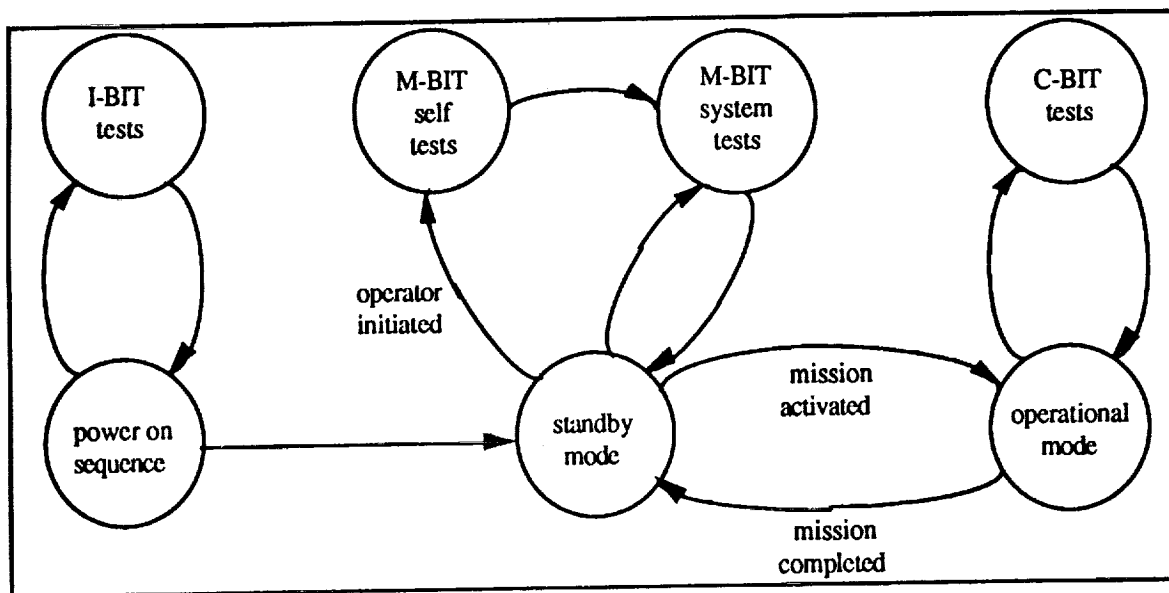


Figure 5-29. System mode and test mode interactions

The I-BIT and the M-BIT are automated sequences of tests which are executed to test the functionality of the AFTA components. In the I-BIT mode a basic set of tests is executed where the primary goal is to ensure the correct operation of all components which are configured into the operational system. Although the M-BIT mode is somewhat similar, philosophically the intent is to extensively test for line-maintenance reasons. The I-BIT is initiated automatically at power-on whereas the M-BIT is commanded by an operator. Because the power on sequence is constrained by time the set of tests comprising the I-BIT suite is a subset of the M-BIT test suite. The C-BIT tests are a set of low-overhead tests which execute during mission critical operations to identify and disable faulty components and to uncover latent faults.

5.6.2. Off-Line Fault, Detection, Isolation and Recovery

In actuality, the functions encompassing fault detection, isolation and recovery are divided into 3 groups – those diagnostic functions performed by the individual components (Off-Line FDIR), those functions performed by a single virtual group in monitoring itself (Local FDIR) and those functions of the system manager which monitor the system components globally (System FDIR).

After a system reset occurs or power is applied to the AFTA components, all components operate individually rather than systematically as a fault tolerant computer. During this phase of operation the Off-Line FDIR exercises a sequence of diagnostic tests of the individual components to determine which components shall be incorporated into the initial configuration of the AFTA.

5.6.3. Local Fault Detection, Isolation and Recovery

After the Network Elements have synchronized with each other the AFTA operates as a fault tolerant system which provides fault tolerant communication mechanisms to processing entities referred to as virtual groups. Using these communication mechanisms each virtual group will exercise some level of fault detection and identification (FDI) capabilities for identification of failures among its processors. Simplex virtual groups may perform only processor self testing. Fault masking groups which are virtual groups consisting of 3 or 4 members can not only perform various levels of testing (unlike simplexes) but can also unequivocally diagnose a failure in a constituent processor. The fault masking virtual group maintains correct operation even when one of its members has failed. Furthermore, it may initiate certain recovery options.

5.6.4. System Fault Detection, Isolation and Recovery

The local FDIR function executing in a virtual group monitors itself and performs some recovery operation which directly affects itself. However, in order to monitor the AFTA system globally and also to determine the health of shared components such as the network elements, a system FDIR is necessary. The system FDIR executes on a single fault masking group and is responsible for high level testing of the AFTA such as a poll of all virtual groups within the system. This is particularly important when a simplex virtual group exhibits faulty behavior. Since a simplex cannot mask itself out of the system configuration via configuration table updates, the system FDIR assumes this responsibility. In addition, some recovery options require global information regarding system resources; this information is unavailable to the local FDIR functions.

The system FDIR function is only one of many system-wide functions of the system manager.

5.6.5. Operational Modes

The AFTA operations are characterized by 2 distinct modes of operations. When the AFTA components are initially powered on or when a reset occurs, all AFTA components are operating independently. The processors on an FCR backplane bus can only communicate with other devices which occupy slots on this bus. The Network Elements in the AFTA are not synchronized with each other; nor are they performing fault tolerant message exchanges. During this mode of operation the AFTA is capable of performing only non-fault tolerant operations. On the other hand, when the Network Elements become synchronized and are capable of performing fault tolerant message exchanges, the AFTA is transformed into a fault tolerant system.

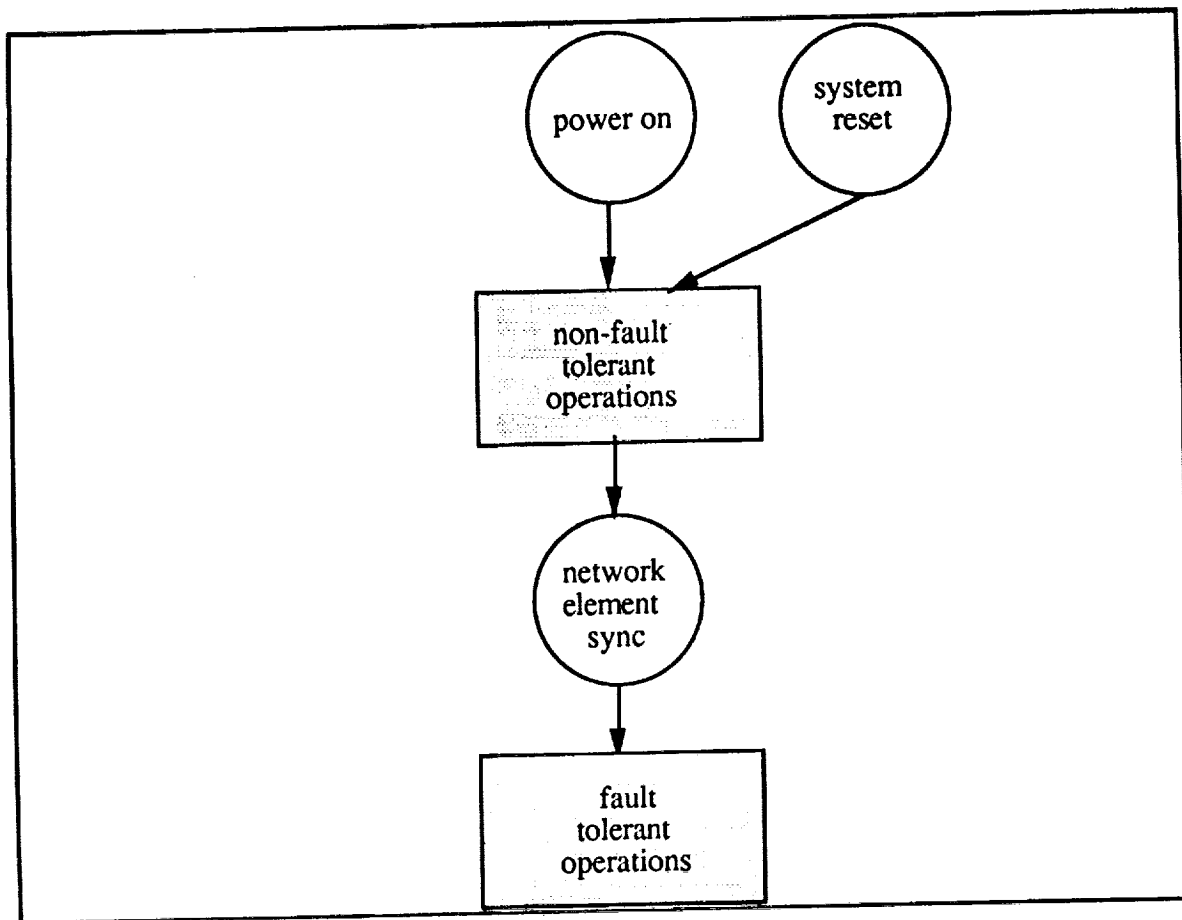


Figure 5-30. Operational modes

The off-line FDIR task is solely responsible for all testing activity while the AFTA is operating as a non-fault tolerant computing system. During fault tolerant operation, both the local FDIR and the system FDIR tasks share responsibility for execution of all testing and recovery functions.

5.6.6. Fault Detection Mechanisms

The AFTA is a highly reliable system which achieves its reliability by exploiting the testing capabilities available in both modes of operation. During non-fault tolerant operations the AFTA executes device self tests which extensively test the functional subcomponents of the device. These tests directly exercise the functionality of a component. If the component behavior disagrees with the expected result the tested component is identified as faulty. These tests are intended to identify faults in a line replaceable module (LRM) with the emphasis on isolating the fault to a chip-level component. This goal can be achieved using on-board diagnostic mechanisms or functionally equivalent tests.

While the AFTA is operating in a fault tolerant state, the repertoire of tests changes to include tests utilizing the inherent fault detection mechanisms. In addition, during this latter operational mode the operating system characteristics also change; a rate group task scheduling mechanism is activated. Consequently, certain other mechanisms become available for exploitation.

Although these 2 modes appear to require disjoint sets of test, this is not the case. When operating in the non-fault tolerant mode, only the device self tests may be exercised. However, during fault tolerant operations, system tests are exercisable and some of the device self tests may be executed provided that they do not violate the operational requirements of the AFTA operating system. These constraints will be discussed in subsequent sections.

5.6.6.1. Enumeration of Mechanisms

The various tests are able to identify faults in the following AFTA components – processors, Network Elements, I/O devices, FCR backplane bus, power conditioners and mass memory devices.

5.6.6.1.1. Processor Self Tests

The processor self test suite will exercise various components of each processing element. Specifically, the tests will exercise the CPU, cache, memory, real-time clock, memory management unit, floating point coprocessor as well as any on-board I/O functions. The following tests define the test suite for the Motorola MVME147 single board micro-computer which will probably be used in the AFTA Brassboard. If a different processor is selected for incorporation into the AFTA a functionally similar set of tests would comprise the processor self tests.

These following set of tests are executed by a processor on its own constituent components.

5.6.6.1.1.1. CPU Tests

Register - The register test performs a thorough test of all registers.

Instruction Set - This test performs various data movements, integer arithmetic, logical, shift and bit manipulation functions.

Addressing Modes - This tests the various addressing modes.

Exception Processing - This tests many of the exception processing functions.

5.6.6.1.1.2. Cache Tests

Basic Data Caching - This tests the gross functionality of the data cache.

- D Cache Tag RAM - This tests the tag RAM by causing accesses to locations generating a variety of tags.
- D Cache Data RAM - This tests the data RAM by causing various values to be written and read from the data cache.
- D Cache Valid Flags - This test verifies that the valid flags are properly set when the associated entry is valid and cleared when the cache is flushed or the individual entry is cleared.
- D Cache Burst Fill - This tests the burst fill mechanism.
- Basic Instruction Caching - This tests the basic functionality of the instruction cache.
- Unlike Instruction Function Codes - This tests the ability of the cache to recognize instruction function codes.
- I Cache Disable - This tests the ability to enable/disable the instruction cache.
- I Cache Invalidate - This tests the ability to invalidate cache entries.

5.6.6.1.1.3. Memory Tests

- Marching Address - This tests the address lines for "stuck high" or "stuck low" conditions.
- Marching One - This tests each RAM location's ability to maintain a single bit in all bit positions.
- Refresh - This tests the refresh mechanism by writing a pattern into RAM and checking it after a time period has elapsed.
- Random Byte - This tests byte data transfer and comparison operations on RAM locations.
- Program - This tests the RAM's ability to execute a self test program in RAM.
- TAS - This tests the Test and Set operation.
- Brief Parity - This tests the parity checking ability on longwords.
- Extended Parity - This tests the parity checking ability on bytes.

5.6.6.1.1.4. MMU Tests

- Root Pointer Register - This tests the root pointer register with a marching bit test.
- Translation Control Register - This tests the translation control register by clearing and then setting the Initial Shift field.
- Super_Prog Space - This test enables the MMU and initiates a table access in supervisor program space.
- Super_Data Space - This tests enables the MMU and initiates an access in supervisor data space.
- Write/Mapped-Read Pages - This tests the ability of the MMU to read data which had been written while the MMU was disabled.

Read Mapped ROM - This tests some of the upper MMU address lines by attempting to access ROM.

Fully Filled ATC - This tests the address translation cache by verifying that all entries in the translation cache can hold a page descriptor.

User_Prog Space - This tests the function code signal lines into the MMU by accessing user program space.

User_Data Space - This tests the function code signal lines into the MMU by accessing user data space.

Indirect Page - This tests the ability of the MMU to handle an indirect descriptor.

Page-Desc Used-Bit - This tests the ability of the MMU to set the Used bit in a page descriptor when the page is accessed.

Page-Desc Modify-Bit - This tests the ability of the MMU to set the Modify bit in a page descriptor when the page is written.

Segment-Desc Used-Bit - This tests the ability of the MMU to set the Used bit in a segment descriptor when the corresponding segment is accessed.

Invalid Page - This tests the ability of the MMU to detect an invalid page and generate a bus error when access is attempted to that page.

Invalid Segment - This tests the ability of the MMU to detect an invalid segment and generate a bus error when access is attempted to that segment.

Write-Protect Page - This tests the page write protect mechanism in the MMU.

Write-Protect Segment - This tests the segment write protect mechanism in the MMU.

Upper-Limit Violation - This tests the capability of the MMU to detect when a logical address exceeds the upper limit of a segment.

Lower-Limit Violation - This tests the capability of the MMU to detect when a logical address exceeds the lower limit of a segment.

Prefetch on Invalid-Page Boundary - This tests determines if the MC68030 rightfully ignores a bus error that occurs as a result of a prefetch into an invalid page.

Modify-Bit and Index - This tests the capability of the MMU to set the Modify bit in a page descriptor of a page which has an index field greater than 0 when the page is written.

Sixteen-Bit User-Program Space - This tests the capability of the MMU to access user program space in 16-bit mode.

Sixteen-Bit Page-Desc Modify-Bit - This tests the ability of the MMU to set the Modify bit in a page descriptor when the page is written in 16-bit mode.

Sixteen-Bit Indirect Page - This tests the ability of the MMU to handle an indirect descriptor in 16-bit mode.

RMW Cycle - This test performs the Test-and-Set instruction in 3 modes to verify that the MMU functions correctly during read/modify/write cycles.

5.6.6.1.1.5. I/O Tests

Ethernet LANCE Chip - This performs an initialization and both internal and external loopback tests on the local area network components.

Z8530 Serial I/O Chip - This tests the functionality of the Z8530 chips for serial transmission and reception.

Interval & Watchdog Timers - This tests the functionality of the interval and watchdog timers.

DMA Controller - This tests the functionality of the DMA device registers.

Power Fail & Bus Error Interrupt Enables - This test writes and reads the AC fail interrupt control and Bus error interrupt control registers.

VMEBus Interface - This tests the VME gate array registers by reading and writing from the local processor bus.

5.6.6.1.1.6. Miscellaneous Tests

Real-Time Clock/BBRAM Test - This tests the real time clock functionality and the battery backed-up RAM.

Bus Timeout Error Test - This tests the local bus time-out and global bus time-out error conditions.

Floating Point Coprocessor Test - This tests the functionality of the floating point coprocessor.

5.6.6.1.2. Network Element Self Tests

The following tests are executed by a processor communicating with the tested Network Element via the FCR backplane bus. These tests exercise various components of the network element:

5.6.6.1.2.1. Processor-Network Element Interface

Dual port RAM - This tests the ability of the Dual port RAM to be written to and read from the processor.

Ring buffer management - This tests the activation of packet transfers and tests the ability of the Ring Buffer Manager to access the proper input and output ring buffers and to check the proper assertion of Output Buffer Full (OBF) and Input Buffer Empty (IBE).

Packet receive interrupt - This tests the functionality of this interrupt mechanism.

5.6.6.1.2.2. Network Element Data Paths

- Class 1 data path FIFO test - This tests the functionality of the data paths through the data path FIFOs using the voting rules for class 1 exchanges.
- Class 2 data path FIFO test - This tests the functionality of the data paths through the data path FIFOs using the voting rules for class 2 exchanges.
- Voter error detection capability - This tests the error detection capability of the voter.
- Message reflection multiplexer - This tests the special data paths involved in source congruent exchanges.

5.6.6.1.2.3. Network Element Global Controller

- Global controller - This tests the functionality of the global controller.
- ISYNC test - This tests the ability of the global controller to achieve synchronization with the other channels using the debug wrap mode.
- Transient NE recovery test - This tests the ability of the global controller to resynchronize with the other channels and update the configuration table.

5.6.6.1.2.4. Scoreboard

- Message class test - This tests the operation of the scoreboard in sending packets of every allowable class.
- Configuration Table Updates - This tests the ability to regenerate the system configuration and to reset all timeouts.
- OBNE Timeout detection - This tests the detection of the OBNE condition and the generation of the OBNE timeout syndrome.
- IBNF Timeout detection - This tests the detection of the IBNF condition and the generation of the IBNF timeout syndrome.
- Scoreboard vote error detection - This tests the detection of a scoreboard vote condition and the generation of the scoreboard vote syndrome.

5.6.6.1.2.5. Inter-Fault Set Communication Links

- Optical data links and TAXIs - This tests the correct operation of the devices used in the optical communication network.

5.6.6.1.2.6. Voted Reset

- Voted reset - This tests the ability to detect a system reset sent by a majority of other Network Elements and to issue a system reset of its own FCR.

5.6.6.1.2.7. Fault Tolerant Clock

- Fault tolerant clock - This tests the ability to detect a self-ahead or self-behind condition and to compensate correctly for this clock skew.

5.6.6.1.3. FCR Backplane Bus Self Tests

The FCR backplane bus will be tested using a standardized suite of self tests to exercise such functions as bus arbitration, bus master control, etc.

5.6.6.1.4. Input/Output Device Self Tests

Input/Output devices may range from a simple "dumb" I/O device to an intelligent device which behaves as a processor. In the former case, a processor on the FCR backplane bus will exercise a suite of tests to evaluate its functionality; in the latter case, the I/O device itself may be capable of executing processor-like self tests.

The tests to exercise the I/O device functions will be determined as I/O devices are identified in subsequent phases of the AFTA program.

5.6.6.1.5. Power Conditioner Self Tests

The power conditioners in each fault containment region will be nominally tested via the on-board set of tests of an intelligent power conditioner.

5.6.6.1.6. Mass Memory Self Tests

The mass memory device is a memory unit with error detection and correction capability consisting of both non-volatile RAM and ROM. It is accessible by all components in the fault containment region via the FCR backplane bus.

The mass memory devices in each fault containment region will be nominally tested via a suite of tests intended to ensure that the memory contents are correct and that the memory addressability is operating properly. In fact, the same memory tests described for the processor on-board memory may be executable on the processor but access the mass memory device if the mass memory exhibits the appropriate characteristics (for example, support for parity). Consequently, the mass memory tests would include the marching address, marching one, refresh, random byte, test-and-set, brief parity and extended parity tests.

5.6.6.1.7. System Tests

The tests discussed previously exercise the functionality of the individual line replaceable modules. Conversely, the system tests exercise functions requiring multiple components operating in tandem to effectively test the system. Because the AFTA is designed as a fault tolerant system, fault detection mechanisms are built into the specially designed interconnection network and are exercised at every message exchange to provide high coverage of faults with low fault latency. The goal of the system self tests is to test the AFTA as an operating entity exercising these fault tolerant mechanisms.

Fault tolerance in the AFTA is implemented using hardware redundancy. A specially designed set of Network Elements operate in tight synchrony to implement fault tolerant message exchanges among processors grouped into redundant virtual groups. The constituent processors in a virtual group communicate with the members of its virtual group and with other virtual groups by synchronously sending messages via the network elements. The Network Elements perform fault tolerant specific operations on messages and deliver voted messages to all members of the destination virtual group. The voting process generates a consistent voted copy of the message as well as error syndrome data which are reported with the delivered message. This error syndrome information can be used to identify faulty components.

A number of tests can be constructed based upon these fault tolerant capabilities as well as upon the characteristics of the operating environment:

- 1) The *presence test* is a means of polling various components to determine if each is active and synchronized. Within the AFTA the presence tests can be employed at 2 levels of abstraction: presence test on members of a virtual group (intra-virtual group) and presence tests on each virtual group in the AFTA (inter-virtual group). The failure of either type of presence test implies that the tested entity is not synchronized.
- 2) The *error syndrome data* described in Section 4 indicates the Network Element detected an erroneous condition. The analysis of this syndrome data can identify either a processor or a Network Element as faulty.
- 3) Because a voted exchange of information generates a consistent message, the voted message mechanism can be used create a consistent voted copy of memory. In this test (called *RAM scrub*) the contents of RAM locations known to have congruent data are compared across all channels by voting the contents of memory. If there is a discrepancy, the fault is logged and the correct value is written to the faulty RAM location.
- 4) In the *PROM check* test the contents of PROM are verified by summing all locations and comparing the results against a voted value.
- 5) The *voter test* will test the Network Element voting mechanism by seeding non-congruent values selectively on each channel of a fault masking group. Not only does this test the composite data but also the syndrome generation.
- 6) The *class test* will test the Network Element voting mechanism by requesting a non-congruent message exchange class selectively on each channel of a fault masking group. This tests the SERP processing and the timeout syndrome generation.

7) The *Network Element presence test* checks the status of each Network Element. Specifically, it determines the synchronization of all Network Elements. This test is used only during the I-BIT mode to determine the initial system configuration.

8) In order to overcome possible software or hardware errors which disable the system the *watch dog timer* is implemented. A timer is periodically reset by software at regular intervals. This timer is decremented periodically by an interrupt process. A timer decremented to 0 is indicative of an error since the timer had not been reset with the predefined time.

9) *Exception handlers* are provided to handle undesirable events such as a divide by zero exception, an illegal instruction or an overflow. In some cases these events are expected by an application and the application should provide a means to account for this situation. However, for those unforeseen situations where a fault causes the trap invocation, a handler will be provided which will initiate remedial action to recover from the fault.

5.6.6.2. Operational Constraints of Fault Detection Mechanisms

During each system mode the time constraints, the requirements on maintenance of mission critical information and even the system configuration differ. Consequently, the tests executed during each of the test modes vary based upon these factors. Because the power on sequence describes the transitions among these operational environments, a brief description of the power on sequence with the emphasis on testing follows (refer to Figure 5-31):

1) Upon initiation of power or manual system reset, the AFTA system is essentially established in an initial, unsynchronized state where each component is operating independently. While in this unsynchronized state, the primary emphasis is to execute as many device self tests as possible. The processors test themselves; subsequently, a single processor is selected which exercises I-BIT self tests of the FCR backplane bus, the Network Element, power conditioner, mass memory and I/O devices.

2) The initial synchronization of the Network Elements is a process whereby the network elements synchronize and commence fault tolerant message exchanges. Each network element, operating independently, can be directed to synchronize upon direction of a processor within the fault containment region or by its own global controller. Subsequent to the initial synchronization, the processors (now members of virtual groups) are capable of performing fault tolerant message exchanges with each other. When the initial synchronization phase terminates a system configuration has been established. This configuration will consist of fault masking virtual groups commensurate with the minimum dispatch complement.

3) After the initial synchronization, a single redundant virtual group will assume the task as the system manager. This system manager virtual group will request that all virtual groups (whether simplex, triplex, or quadruplex) transfer their diagnostic test results to the system manager. The system manager evaluates the results and reconfigures those redundant virtual groups which contain a faulty component with the intent of achieving the minimum dispatch complement for computing resources at the required reliability level.

4) After the redundant system configuration has been established, the system manager commands each virtual group to initiate the real-time scheduler and to commence the system tests. The minimal set of I-BIT system tests will be exercised.

5) When it has been determined that the minimum dispatch complement for the current mission has been established, the AFTA system will be established in a state referred to as "operational standby". During this state the comprehensive suite of M-BIT system tests will be exercised until the mission is activated.

6) When the mission is activated, the system is established in the "mission critical" mode where the C-BIT tests are executed concurrent with the mission functions. As indicated in Figure 5-31, the AFTA operating system prevents execution of any BIT other than C-BIT until a reliable indication is given that the mission is over and it is safe to enter other diagnostic modes. This could be a composite indication from mutually corroborative sources such as the weight-on-wheels switch, rotor RPM, vehicle INS and rate gyro system, propulsion status, and pilot discrete(s).

7) Alternatively, from the standby state an operator may command the M-BIT sequence of tests which execute similar to the I-BIT tests.

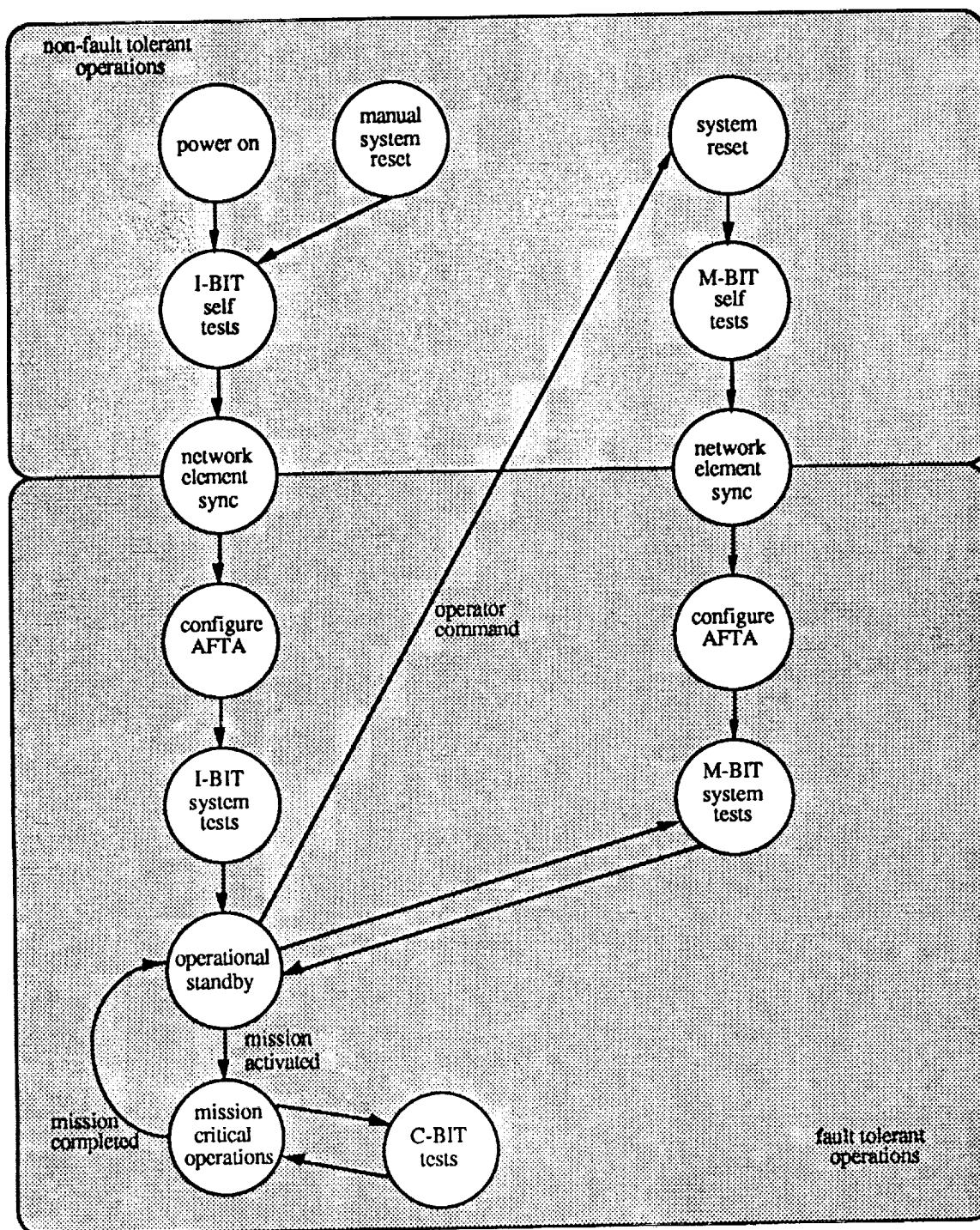


Figure 5-31. Test Mode Sequences

5.6.6.2.1. I-BIT Mode Self Tests

The I-BIT test mode is automatically initiated when power is applied to all AFTA components. However, the I-BIT test mode is constrained by a requirement that this mode be active for only seconds. Presumably, shortly after initiation, it is desirable that the vehicle

be mission ready. Because of this time limitation only a subset of the self tests employed in the M-BIT mode shall be executed. The emphasis in the selection of I-BIT tests is primarily to exercise the functionality of all LRMs in the AFTA and secondly to test those components as comprehensively as time permits.

The component tests will be sequenced such that components deemed to be non-faulty will exercise subsequently tested components. This methodology requires that the initial component test itself. Although it cannot be guaranteed that a faulty initial component will correctly conclude its own health, subsequent system testing can detect faulty behavior and that faulty component will be eliminated.

During the non-fault tolerant operational mode, the AFTA components are not synchronized or operating as a fault tolerant computer. Consequently, there is no critical information which must be maintained aside from test results and there are no synchronization constraints. The tests can destructively change memory locations on the processors, test the Network Elements in a debug wrapback mode, or change the bus master on the FCR backplane bus. In addition, since the real-time operations have not commenced, the scheduling constraints are relaxed.

5.6.6.2.2. M-BIT Mode Self Tests

The M-BIT test mode is initiated by an operator in an operational environment with the expressed purpose of extensively testing all components of the AFTA. Because of the lack of a severe time constraint, the suite of tests could conceivably be the identical set as that executed in the depot test mode. However, there are two primary differences between these test modes – the operator interface and the automation of the test sequence. The site of the M-BIT execution is on-board a hosting vehicle which naturally, implies that the operator is either the vehicle operator or a line maintenance crew. In addition, the M-BIT tests exercise all components of the AFTA as an automated sequence. In contrast, the depot tests are conducted at a remote repair facility by a repair technician exercising a set of tests on a single LRM.

The other operational constraints are the same as those of the I-BIT test mode.

5.6.6.2.3. I-BIT Mode System Tests

The I-BIT test mode is a bridge between a system reset condition and a fault tolerant operational state. Because the time constraint to transition from an initial state to a fully operational environment (that is, operational standby), is so severe, only a minimal set of the system tests is executed. In fact, only an NE presence test is incorporated during this mode to ensure that all Network Elements are operational and synchronized.

5.6.6.2.4. *M-BIT Mode System Tests*

Because of the lack of a severe time constraint (as with the I-BIT) the suite of tests which can comprise the M-BIT test mode can be very extensive. However, as currently envisioned, the M-BIT system tests will comprise the same suite of tests as the C-BIT.

5.6.6.2.5. *C-BIT Mode Tests*

During the C-BIT mode the AFTA is operating in a state where: 1) real-time scheduling is enforced, 2) mission critical operations occur and 3) redundant virtual groups exist. These constraints require that the tests enacted during this mode be unobtrusive.

During the C-BIT mode, the AFTA system performs mission critical tasks within the confines of a real-time scheduler. These constraints pose two requirements for C-BIT testing – 1) information must be preserved and 2) the operation of these tests must be unobtrusive. Consequently, a minimum of computing resources must be consumed in the analyzing the inherent fault detection mechanisms and data integrity must be maintained.

Any device self test implemented as a C-BIT must ensure that it does not modify the mode of operation of any AFTA component in an unrecoverable way. Examples include tests which change the processor status register or activate the memory management unit, tests which cause the Network Elements to desynchronize, or tests which alter bus arbitration on the FCR backplane bus. Because of the operational requirements of the system and because on-board diagnostics typically do not preserve system state, many of the manufacturer supplied set of tests are inoperable in a real-time operational environment. For instance, memory tests typically modify, read, and check memory locations without preserving the information. Therefore, the implementation of the self tests for the C-BIT mode will be different than those for the other test modes.

In addition, because the system configuration could conceivably be a mixed redundancy system consisting not only of fault masking groups but of simplexes as well, the system tests must ensure that all virtual groups are active and that all are operating properly.

5.6.6.3. *Mapping of Fault Detection Mechanisms to Test Modes*

Because the goal of the AFTA design is to create a digital computing system of the highest possible fault coverage, it is imperative that a comprehensive set of tests be executed during all test modes. Since the time constraints and system configuration vary for each test mode, the suite of tests for each test mode will differ. In fact, the suite of tests will be a composite of both the self tests and the system tests whenever practical. For instance, it is impossible to execute a system test when the hardware is operating in non-fault tolerant mode. On the other hand, a self test may require altering some datum (for exam-

ple, a status register) which allows for the possibility of a catastrophic change of state which jeopardizes mission critical information.

The following series of tables delineate the AFTA tests. Because some tests are executable only in either one of the operational modes whereas others can be executed in both modes, the individual tests are marked for each test mode as follows:

- 1 test is executable in non-fault tolerant mode only
- 2 test is executable in fault tolerant mode only
- 3 test is executable in both modes

For those tests which are executable in both modes the actual implementations of the test could be different although they functionally perform the same operations.

5.6.6.3.1. Processor Self Tests

CPU Tests:	depot	I-BIT	M-BIT	C-BIT
Register	1	1	3	3
Instruction Set	1	1	3	3
Addressing Modes	1	1	3	3
Exception Processing	1	1	1	

Cache Tests:	depot	I-BIT	M-BIT	C-BIT
Basic Data Caching	1	1	1	
D Cache Tag RAM	1		1	
D Cache Data RAM	1		1	
D Cache Valid Flags	1		1	
D Cache Burst Fill	1		1	
Basic Instruction Caching	1	1	1	
Unlike Instruction Function Codes	1		1	
I Cache Disable	1		1	
I Cache Invalidate	1		1	

Memory Tests:	depot	I-BIT	M-BIT	C-BIT
Marching Address	1	1	3	3
Marching One	1	1	3	3
Refresh	1		1	
Random Byte	1	1	3	3

Program	1		1	
TAS	1	1	3	3
Brief Parity	1	1	1	
Extended Parity	1	1	1	

MMU Tests:	depot	I-BIT	M-BIT	C-BIT
Root Pointer Register	1	1	1	
Translation Control Register	1	1	1	
Super_Prog Space	1		1	
Super_Data Space	1		1	
Write/Mapped-Read	1		1	
Read Mapped ROM	1		1	
Fully Filled ATC	1		1	
User_Prog Space	1		1	
User_Data Space	1		1	
Indirect Page	1		1	
Page-Desc Used-Bit	1		1	
Page-Desc Modify-Bit	1		1	
Segment-Desc Used-Bit	1		1	
Invalid Page	1		1	
Invalid Segment	1		1	
Write-Protect Page	1		1	
Write-Protect Segment	1		1	
Upper-Limit Violation	1		1	
Lower-Limit Violation	1		1	
Prefetch on Invalid-Page Boundary	1		1	
Modify-Bit and Index	1		1	
Sixteen-Bit User-Program Space	1		1	
Sixteen-Bit Page-Desc Modify-Bit	1		1	
Sixteen-Bit Indirect Page	1		1	
RMW Cycle	1		1	

I/O Tests:	depot	I-BIT	M-BIT	C-BIT
Ethernet LANCE Chip	1	1	1	
Z8530 SIO Chip	1		1	
Interval & Watchdog Timers	1	1	1	
DMA Controller	1		1	

Power Fail & Bus Error Interrupt Enables	1		1	
VMEBus Interface	1	1	1	

Miscellaneous Tests:	depot	I-BIT	M-BIT	C-BIT
Real-Time Clock/BBRAM Test	1	1	1	
Bus Timeout Error Test	1	1	1	
Floating Point Coprocessor Test	1	1	3	3

5.6.6.3.2. Network Element Self Tests

Processor-Network element interface:	depot	I-BIT	M-BIT	C-BIT
Dual port RAM	1	1	1	
Ring buffer management	1	1	1	
Packet receive interrupt	1	1	1	

Network element data paths:	depot	I-BIT	M-BIT	C-BIT
Class 1 data path FIFO test	1	1	1	
Class 2 data path FIFO test	1	1	1	
Voter error detection capability	1	1	1	
Message reflection multiplexer	1	1	1	

Network element global controller:	depot	I-BIT	M-BIT	C-BIT
Global controller	1	1	1	
ISYNC test	1	1	1	
Transient NE recovery test	1	1	1	

Scoreboard:	depot	I-BIT	M-BIT	C-BIT
Message class test	1	1	1	
Configuration table updates	1	1	1	
OBNE timeout detection	1	1	1	
IBNE timeout detection	1	1	1	
Scoreboard vote error detection	1	1	1	

Inter-fault set communication links:	depot	I-BIT	M-BIT	C-BIT
Optical data links and TAXIs	1			

Voted reset:	depot	I-BIT	M-BIT	C-BIT
Voted reset	1		1	

Fault tolerant clock:	depot	I-BIT	M-BIT	C-BIT
Fault tolerant clock	1	1	1	

5.6.6.3.3. FCR Backplane Bus Self Tests

	depot	I-BIT	M-BIT	C-BIT
TBD	1	1	1	
TBD	1	1	1	
TBD	1	1	1	

5.6.6.3.4. Input/Output Device Self Tests

	depot	I-BIT	M-BIT	C-BIT
TBD	1	1	3	3
TBD	1	1	3	3
TBD	1	1	3	3

5.6.6.3.5. Power Conditioner Self Tests

	depot	I-BIT	M-BIT	C-BIT
TBD	1	1	1	
TBD	1	1	1	
TBD	1	1	1	

5.6.6.3.6. Mass Memory Self Tests

Memory Tests:	depot	I-BIT	M-BIT	C-BIT
Marching Address	1	1	3	3
Marching One	1	1	3	3
Refresh	1		1	
Random Byte	1	1	3	3
TAS	1	1	3	3
Brief Parity	1	1	1	
Extended Parity	1	1	1	

5.6.6.3.7. System Tests

	depot	I-BIT	M-BIT	C-BIT
Intra-virtual group presence test			2	2
Inter-virtual group presence test			2	2
Syndrome analysis			2	2
RAM scrub			2	2
PROM check			2	2
Voter test			2	2
Class test			2	2
NE presence test		2		
Watchdog timer			2	2
Exception handlers		2	2	2

5.6.7. Fault Diagnosis

During both non-fault tolerant and fault tolerant operations the AFTA system performs various levels of testing commensurate with the operational constraints. Because of the operational environment and the ultimate goal of comprehensively testing the AFTA during all operations, the tasks of testing and result analysis is divided among the three FDIR functions – Off-Line FDIR, Local FDIR and System FDIR. This section describes the overall methodology used by each task and the self and system tests implemented by each.

5.6.7.1. Non-Fault Tolerant Operations

During the non-fault tolerant mode the emphasis is on ensuring that the constituent components of the AFTA are operating correctly. This is accomplished by exercising functional components of each LRM in the AFTA with a series of diagnostic level tests.

Off-Line FDIR is initiated when the system is reset at which time the AFTA components are operating independently. In other words, the Network Elements are not synchronized, the processors act as individual processors rather than as members of a virtual group and the I/O devices are in an initial state.

Off-Line FDIR systematically sequences through a series of diagnostic level self tests to exercise all AFTA components. There are 2 distinct series of tests corresponding to the I-BIT and M-BIT modes. These sets were necessitated because of the constraints regarding the amount of time allotted to perform testing in these modes. The enumeration of the tests comprising each test mode can be deduced from the tables in the fault detection mechanisms section.

Procedurally, Off-line FDIR initiates a series of self tests of each processor in the system. As each processor completes its suite of tests and is determined to be non-faulty, it claims an area of the dual ported RAM as its interface with the Network Element. (Section 4 describes the memory map and the functions of this RAM.) The first non-faulty processor in each fault containment region shall be responsible for testing the network element, FCR backplane bus, power conditioner, mass memory, and I/O devices. All fault information will be saved by each processor in the FCR's non-volatile mass memory for later dissemination.

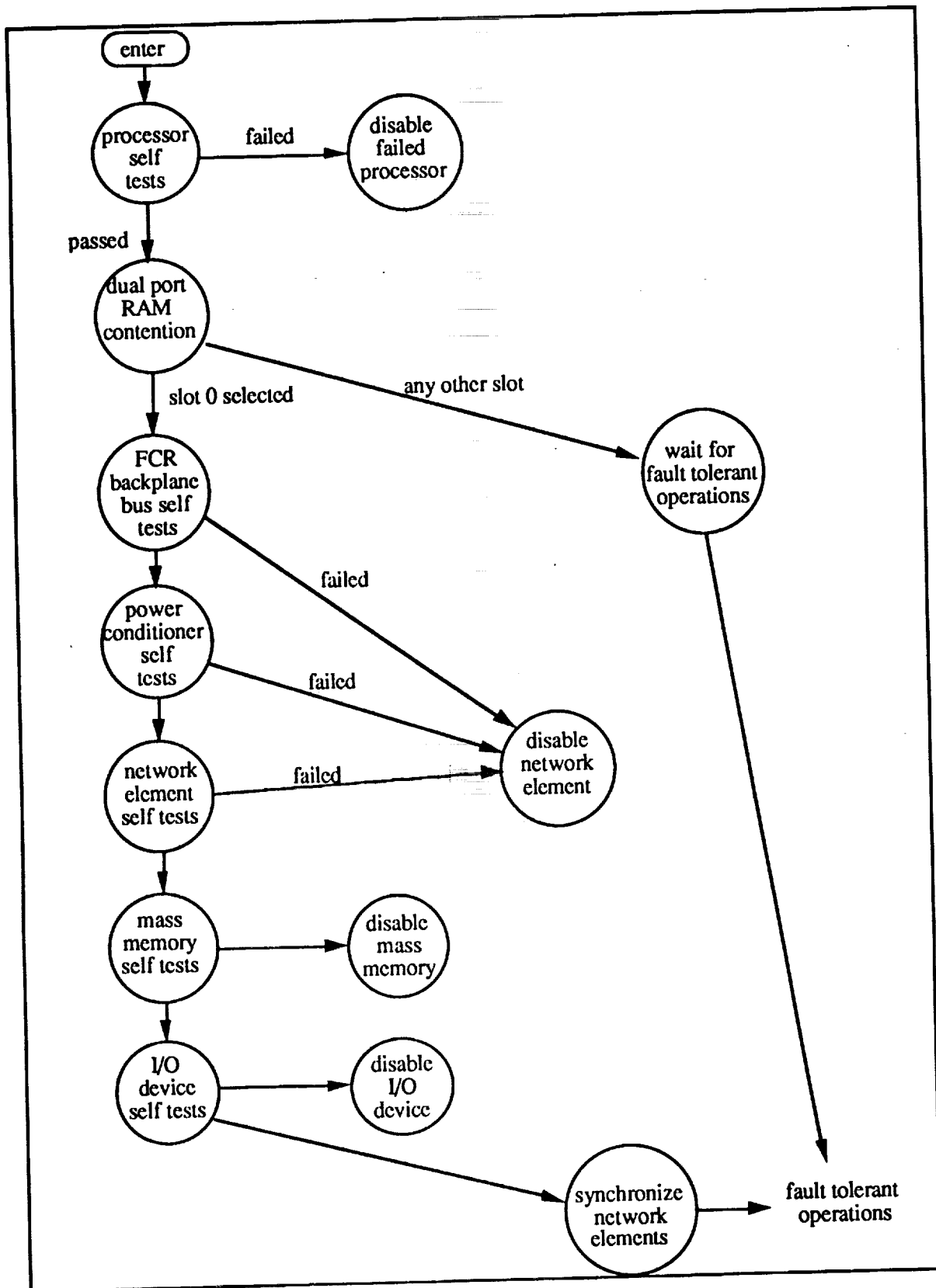


Figure 5-32. Off-Line FDI Overview

Off-Line FDIR uses the device self tests exclusively for diagnosis of faulty components. Because these tests directly exercise the functionality of LRMs the diagnosis of a specific LRM as faulty is obviously trivial. However, since the self tests exercise the functionality of an LRM, the failure of a test identifies not only the failed LRM but also the failed LRM function which maps to a chip or set of chips. This chip level diagnostic information is retained for dissemination to a maintenance crew.

5.6.7.2. Fault Tolerant Operations

When the AFTA system hardware is operating synchronously with fault tolerant message exchanges, it is capable of exercising the system tests which employ the inherent fault detection mechanisms to provide fault tolerance. In addition, the system is also capable of executing some of the self tests. Because of the requirement to execute unobtrusively during fault tolerant operations, only a subset of the entire suite of self tests can be executed. In particular, only some of the processor self tests can be performed.

Although all test modes execute system tests, only the M-BIT and C-BIT modes utilize the full capabilities of these tests. The system test capability of the I-BIT is minimal.

During fault tolerant operations the duties of fault detection, isolation and recovery are shared between local FDIR and system FDIR. Local FDIR executes on each virtual group and is able to diagnose faults in the constituent processors of that virtual group. System FDIR, on the other hand, executes on a single fault masking group; it diagnoses failures in all other AFTA components.

5.6.7.2.1. Local Fault Detection and Isolation

Each redundant fault masking virtual group executes the intra-virtual group presence test, syndrome analysis, RAM scrub, PROM check, watch dog timer, and provides exception handlers for certain unusual conditions. Of these tests the intra-virtual group presence test and the syndrome analysis are invoked on an iterative basis to check for an unsynchronized channel, a failure in the Network Element hardware, and processor failures. This synchronous test methodology is depicted in Figure 5-33.

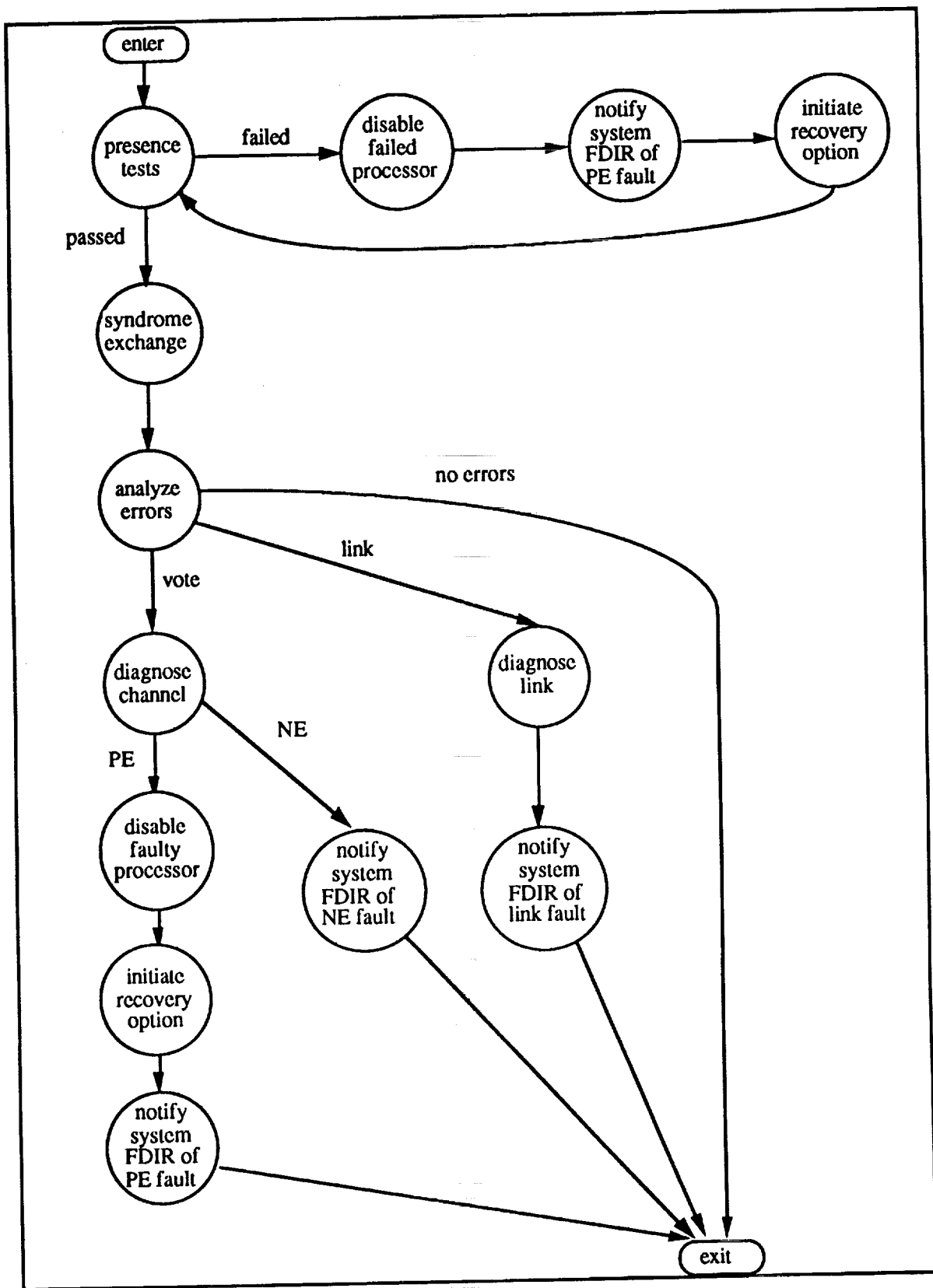


Figure 5-33. Synchronous FDI Overview

Although the local FDI can detect failures in components other than the constituent processors of its virtual group, it is responsible only for the diagnosis and disabling of its member processors. Other component failures such as failures in a Network Element failure or an I/O device are analyzed and disabled by system FDI.

When a processor is identified as being faulty, local FDI disables the faulty processor so that it does not adversely affect operation. Specifically, FDI disables the voted outputs from the faulty processor, reports the failure to the FDI system manager and initiates the selected recovery option.

5.6.7.2.1.1. Intra Virtual Group Presence Test

An unsynchronized processor is detected by means of the intra-virtual group presence test. This test detects an unsynchronized processor by sending a unique pattern from each member of the virtual group via source congruent message exchanges through the network. If the result received is not the expected pattern, the processor originating the exchange is judged not present and, therefore, desynchronized from the other channels. When the synchronized channels detect the loss of synchronization of a processor, the synchronized members of the virtual group disable the faulty channel.

5.6.7.2.1.2. Syndrome Analysis

Failures in the processors of a virtual group or Network Elements are detected by analyzing the error syndrome delivered with message packets by the Network Element hardware. The error syndrome defines both vote errors and link errors generated during handling of the packet by the Network Elements. Vote syndrome are generated during a message exchange if a miscompare is detected by the Network Element hardware during the voting of data received from the redundant channels. Link errors are generated if the receiver fails to detect the transmitter/receiver synchronization pattern. Since the error syndrome is delivered with the message exchanges, the message handling primitives extract this information on a message class basis for analysis by FDI.

The Network Element generates the syndrome just prior to delivery of the message to the processor. Therefore, the syndrome data are non-congruent with the other members of its virtual group. In order to prevent divergence of the synchronous channels which must operate on identical inputs to maintain synchrony, the channels must participate in a series of source congruent exchanges of this syndrome data. Upon completion of the syndrome exchange process, each channel has a copy of all channels' syndrome data.

The syndrome analysis identifies a fault in either (1) a Network Element (including the transmitter-to-receiver link), or (2) a processor which generated incorrect voted data. The analysis occurs in a 3 step process:

- 1) Analyze the vote syndrome,
- 2) Analyze the link syndrome, and
- 3) Correlate vote and link results to identify a faulty component.

The vote syndrome analysis compares the pattern across channels for each message class with known fault patterns. In each case a hypothesis testing methodology is used where a channel is assumed faulty, that channel is masked out and the resultant pattern compared with the known pattern. This analysis results in the indication of a faulty processor associated with a message class.

The link syndrome analysis identifies either a transmitter or receiver as faulty. However, rather than analyzing all channel's syndrome against a specific pattern, each channel's link syndrome is analyzed individually. Essentially, each analysis generates 2 hypotheses – one indicative of the transmitter indicated by the syndrome and the other representing the detecting channel's receiver. The subsequent count of the errors detected verify either one of these hypotheses. Multiple channels detecting the same link error indicates a transmitter fault; a single channel detecting the link error implies the receiver fault.

Because link faults can generate vote errors, it is important to identify the source of a vote error. In the absence of other error syndromes, processor faults are identified as vote errors on voted messages. Any combination of errors which includes a link error are attributable to either a transmitter or receiver fault in the appropriate Network Element. Although Network Element errors may manifest themselves as a vote syndrome on either voted messages or source congruent errors, the diagnosis of a Network Element fault is identified as a vote error on a source congruent message.

The syndrome data analyzed by a virtual group is that data delivered with messages which the virtual group addressed to itself. Consequently, the members of a virtual group diagnose its constituent processors. The virtual group also disables its own faulty processor. Although a redundant virtual group can also diagnose Network Elements as faulty, it merely reports these diagnoses to a system FDI function which performs further analysis and, if necessary, Network Element recovery.

5.6.7.2.1.3. Self Tests

Because the AFTA is designed to withstand a specific number of simultaneous faults, it is imperative that the number of faults be confined to a minimum. Exceeding the number of simultaneous faults could result in total system failure. For this reason, background self tests were devised to minimize the possibility of simultaneous failures by checking for latent faults in the AFTA processors. When a faulty component is uncovered, that compo-

nent can be eliminated from the system configuration thereby, reducing the likelihood of simultaneous faults.

While the majority of the system tests are effective for the identification of processor faults in fault masking groups, the testing (and subsequent diagnosis) of simplex virtual groups is primarily provided by self tests.

The background self tests exercise processor components and comprise a comprehensive as is feasible set of tests which are executed as low frequency background tests. These tests include memory tests (for example, RAM pattern tests) and CPU tests (register, instruction, addressing modes).

Failures detected using these tests will adhere to the transient fault analysis and recovery policies defined in a subsequent section.

5.6.7.2.2. *System Fault Detection and Isolation*

System FDI is responsible for the coordination of system status and fault information as well as for testing and analysis of shared components. Specifically, system FDI will be responsible for

- 1) maintaining the current status of every system component,
- 2) initiating Network Element tests and analyzing test results,
- 3) initiating I/O device tests and analyzing test results,
- 4) evaluating the fault diagnosis information of other fault masking groups with regard to Network Element failures, and
- 5) analyzing syndrome data indicative of Byzantine faults, and
- 6) collecting and reporting of fault information logged in the mass memory devices in each fault containment region.

System FDI will evaluate the status of every system component by executing the inter-virtual group presence test as well as by accepting update status from each system component indicating the faulty component. The *inter-virtual group presence test* is essentially a poll of all virtual groups within the system. Failure of a virtual group to respond to the message within a specific time period is indicative of a fault. This is especially important for the analysis of simplex processors which may have failed without communicating that information to the system manager.

The system Network Element tests include a *voter test* and a *class test*. Each of these tests exercises the Network Element by seeding either non-congruent data or non-congruent class information into the Network Element. This testing will systematically exercise each Network Element. However, because the AFTA system configuration could consist of up to 5 Network Elements whereas the system manager fault masking group may consist of a maximum of 4 members, the system manager itself is incapable of testing all network elements. Instead, it may assign some portion of the test task to another fault masking group.

Although the local FDI functions can detect and identify a Network Element failure, it cannot formally diagnose a Network Element as faulty. This information is sent to the system FDI which may perform additional analysis and actually perform some remedial action.

There may be situations where the syndrome information maintained by the local FDI on a virtual group is inconsistent. Either all members of the virtual group do not concur on the identification of a component or not all members agree that a fault even exists. This type of syndrome information is indicative of a Byzantine fault where the faulty component maliciously communicates some information to a fault containment region and some other data to another fault containment region. Faults of this type would generally be indicative of a faulty Network Element and, hence, must be handled by the system FDI.

5.6.8. Recovery options

When a component has been diagnosed as faulty it will be disabled. The network element hardware has masking capabilities to mask a failed component. For instance, there is a processor mask which can disable a faulty processor's participation in a voted message exchange. Furthermore, a faulted processor can be excluded from a virtual group thereby preventing it from communicating with other virtual groups. In addition, a network element can be masked causing the other Network Elements to ignore data and clock signals from the disabled Network Element. Although these masking capabilities can prevent a faulty component from corrupting information in the other fault containment regions, it is desirable to inhibit the faulty component from affecting other components which share the FCR backplane bus. For this reason the component will be disabled from both a component and a system perspective.

The response to failure section defines the actions of the individual components to limit faulty behavior to the most confining failure envelope. The subsequent section describes from the system perspective the methods to recover from a component failure while efficiently utilizing system resources.

5.6.8.1. Response to Failure of Test

When a component fails it is highly desirable to contain the faulty behavior to the smallest possible extent. Although this faulty behavior is contained within the FCR boundaries, the component could corrupt other devices which share the same FCR backplane bus. The following actions attempt to reduce the failure envelope to include only that device itself. Failure to adequately limit the damage by use of these measures will ultimately lead to the failure of the FCR which is unavoidable under these circumstances.

1) *Processor* – During non-fault tolerant operations a processor may be able to detect itself as faulty. The processor will attempt to log the failure in a mass memory device and to disable itself by executing a reset. During fault tolerant operations the processor itself may diagnose itself as faulty via a processor self test; it may reset itself. Alternatively, the virtual group may diagnose the fault. Depending upon the system recovery strategy and the transient analysis policy the virtual group may generate a voted reset.

2) *Network element* – If a Network Element exhibits faulty behavior, the only remedial action the testing processor can perform during non-fault tolerant operations is to issue a Network Element reset to prevent that Network Element from initially synchronizing with the other Network Elements. During fault tolerant operations, the system manager virtual group which diagnoses Network Element failures may permanently disable the faulty network elements via the Network Element mask and may perform a voted reset.

3) *I/O device* – If an I/O device is declared faulty during non-fault tolerant operations, the testing processor can reset that device and log the fault in the mass memory. The monitor interlock is asserted to disable that I/O device (Refer to Section 4). Although the I/O device may be reset and disabled via a monitor interlock during fault tolerant operation, the specific mechanisms have yet to be defined.

4) *FCR backplane bus* – Since all communication between the Network Element and the attached devices (that is, processors, I/O devices, mass memory) occurs via the FCR backplane bus the only appropriate response to a FCR backplane bus failure is to reset the Network Element. Masking the Network Element disables the faulty component totally. A failure of the FCR backplane bus during fault tolerant mode would be attributable to either a processor or a Network Element because the bus is not directly tested by any test in this mode. Nonetheless, those components exhibiting the faulty behavior will be identified and disabled via a reset.

5) *Power conditioner* – Because a power conditioner regulates the voltage to the entire fault containment region, its failure could generate spurious signals to any component within the fault containment region. Failure to adequately mask this failure could result in

later system failure. Consequently, in order to prevent the possibility of a system failure it is imperative that the Network Element be reset and that Network Element be masked.

6) *Mass memory* – Because the mass memory device is a somewhat passive component, the manner in which it is disabled depends upon the nature of the fault. If a memory location is faulty as in a "stuck-at-one" condition, than the mass memory device can be ignored or the faulty locations bypassed. However, if the FCR backplane interface with the mass memory device is inoperative, the mass memory could disrupt communications across this bus and would require disabling the entire fault containment region via a network element reset.

5.6.8.2. System Recovery

During fault tolerant operations the system tests and the processor self tests may indicate a component as failed. At that time some remedial action must be taken in order to remove that faulty component from the operational system. Because of the dynamic reconfigurability of the AFTA architecture many recovery options are possible ranging from merely masking the faulty component such as a Network Element to integration of a spare processor as a replacement for a faulted one. These recovery actions are different for each type of failed component (that is, processor, network element, or I/O device). Furthermore, when recovery from a failure is initiated, the current system mode is an important factor because it defines the time constraints for execution of a recovery strategy.

As the recovery strategies are discussed the operational requirements of each strategy are addressed. These constraints may or may not be commensurate with the operational requirements of the mission critical environment. Hence, an appropriate recovery strategy should be selected based upon the mission and its system mode. Figure 5-34 depicts rather qualitatively the appropriateness of a recovery methodology given the system mode (that is, power-on, standby or operational).

recovery option \ system mode	system mode		
	power-on	standby	operational
graceful degradation	fairly practical	minimally practical	highly practical
processor resynchronization	highly practical if power-on time sufficient	highly practical if standby time sufficient	highly practical if large minor frame
processor reintegration	highly practical if power-on time sufficient	highly practical if standby time sufficient	highly practical if large minor frame
processor replacement	highly practical if power-on time sufficient	highly practical if standby time sufficient	highly practical if large minor frame
processor replacement with initialization	highly practical if power-on time sufficient	highly practical if standby time sufficient	fairly practical if task restart possible
task migration	highly practical	highly practical	highly practical if task restart possible
network element resynchronization	highly practical	highly practical	highly practical
network element masking	highly practical	highly practical	highly practical

Figure 5-34. Qualitative evaluation of recovery methods

There are two primary criteria for the selection of a recovery option – 1) the operational environment of the system when the fault was uncovered and 2) the type of faulty component. The operational environment criterion defines the system mode of operation and is indicative of the system constraints. These constraints may require that the reconfiguration process complete within a minor frame (for example, 10ms) or that the recovery time can be significantly greater. They may also require that mission critical information be maintained. Obviously, the recovery option is also contingent upon the type of failed compo-

ment. A reconfiguration policy to recover a failed processor is drastically different than that of a failed Network Element.

These recovery options are oriented at system goals as a means of dealing with a failure in a component. Some of the recovery options attempt to reintegrate the failed component to determine if the fault was transient. Other recovery options discard the failed component without an attempt to "recover" the diagnosed component.

The options discussed recover the system from failures in processors and network elements. The options for recovery from an input/output device fault are the same as those for processors if the I/O device has the processor-like functionality to interface to the Network Element as a member of a redundant virtual group. The specification of recovery options for I/O devices will be addressed as I/O devices are selected for incorporation into the AFTA.

5.6.8.2.1. Recovery from Processor Failure

If a processor failed, numerous strategies exist for system recovery. Although it is highly desirable to recover a channel, it may not always be possible because of mission critical constraints. For this reason a number of possible recovery options are posed which have various operating characteristics. Depending upon the mission mode, these characteristics may make a recovery option feasible to execute without irreparable harm to the mission.

5.6.8.2.1.1. Graceful Degradation

During mission critical operations a redundant virtual group tests itself using such tests as the intra-virtual group presence test or syndrome analysis. In the absence of a common mode fault, the virtual group can correctly deduce that a member has failed and can initiate corrective action. Specifically, a virtual group can gracefully *degrade* its redundancy level by issuing a configuration table update message which eliminates the faulted channel. The CT update message can reconfigure a redundant group and create a simplex atomically eliminating the requirement that each virtual group in the system be cognizant of an upcoming system reconfiguration. This has the net effect of initiating and terminating a reconfiguration within a minor frame.

One disadvantage to this alternative is that the redundancy of the virtual group is decreased. A quadruply redundant group would degrade to a triplex; a triply redundant virtual group would become a degraded triplex which is essentially a triply redundant group which has a single channel's voted messages masked out. The faulted channel's data cannot contribute in any voted message. Operating as a degraded triplex is undesirable because its performance may be significantly penalized if the faulted channel fails to respond

to message requests and timeout penalties are sustained by the degraded triplex. A simplex, of course, cannot be affected by this recovery technique.

5.6.8.2.1.2. Processor Resynchronization

When a virtual group member is judged to have lost synchronization with the other channels of its virtual group, the resynchronization recovery strategy attempts to resynchronize that lost channel and reintegrate it in order to maintain the redundancy level of the virtual group.

A processor which fails the intra-virtual group presence test is deemed to have lost synchronization and attempts to resynchronize itself with the other channels. The failed channel itself detects the failure, primarily via a watchdog timer mechanism; the synchronized channels detect the failure when the channel fails to respond to its presence test. When resynchronization (also referred to as lost channel synchronization) has been achieved, the state of the failed processor (now resynchronized) must be made congruent with the other synchronized processors. This is accomplished by an alignment process. This is a process whereby the machine state of all members of a virtual group become congruent by voting all congruent memory, registers, and timers.

There are essentially two participants in the resynchronization process – the lost channel and the synchronized channels. When the lost channel detects loss of synchronization with the other members, it immediately invokes a lost channel synchronization procedure. The synchronized channels periodically invoke this procedure when the transient analysis function deems it appropriate to attempt recovery of a failed channel.

The resynchronization function consists of two control streams – one for the lost channel and the other for the synchronized channels. These are depicted in Figure 5-35. The lost channel executes a `pickup_sync` routine which essentially listens to incoming messages for the specific *pickup* message. When it detects this message, the lost channel participates in the resynchronization presence test. Conversely, the synchronized channels perform a voted message exchange of the pickup message. Subsequently, these channels execute the resynchronization presence test which, like the presence test, consists of a series of source congruent message exchanges. These message exchanges send source specific patterns which differ from those used in the presence test. For each exchange, a comparison is made of the pattern received for the given exchange against the pattern expected for a successful exchange. A match indicates that the given channel is operating in synchronism with the channel sourcing the exchange; a mismatch indicates a lack of synchronism. The lost channel returns to `pickup_sync` if it failed to synchronize; the synchronized channels return to the scheduler if the lost channel failed to resynchronize.

Because the lost channel had been desynchronized with the other channels from some time period, the processor state in the lost channel is most likely different than the processor state of the synchronized channels. It is imperative that once synchronous operation among all channels is established, the processor state must be made congruent across channels such that synchronous operation will continue. This ensures that the control flows in each channel are synchronous. Consequently, before normal scheduling resumes, the channels must

- 1) align their congruent memory and
- 2) align their clocks.

The memory alignment is a process whereby RAM and registers in each processor becomes congruent. In this alignment process each processor within the channel transmits and receives voted messages representing blocks of its congruent memory areas. Voting this series of messages via the Network Element voted message mechanisms creates bit-wise voted copies of the memory across all channels. Because the goal is to have identical state across all channels, the alignment process also includes the processor registers.

In each channel the local interrupt timer is responsible for generating the minor frame interrupt, typically every 10ms. Because synchronous channels reset their local interrupt timers in tandem, each channel congruently maintains system time. When a channel is desynchronized it does not participate in the time synchronization; its system time tends to drift. The time alignment process restarts congruent time keeping by resetting and activating its local clock after a lost channel has been resynchronized and its state aligned. Because the memory alignment process (through message exchanges) has tightly synchronized each channel, the clock activation minimizes the skew among the timer interrupts.

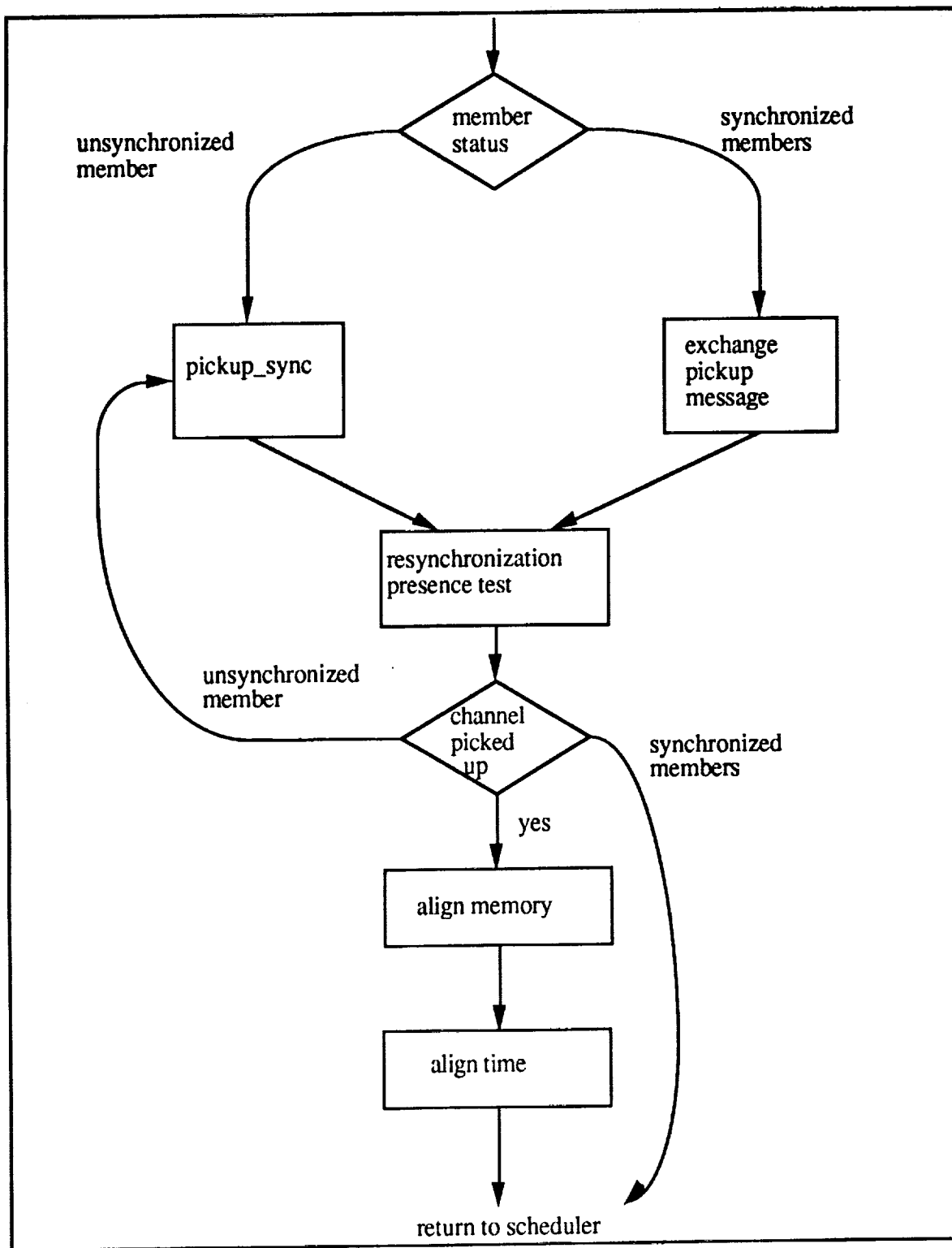


Figure 5-35. Lost Channel Synchronization

Although this recovery option is an attractive alternative because it maintains the redundancy level of the virtual group, it is not suitable to all operational situations because the

alignment process requires exclusive control of the virtual group for a significant time period (on the order of 1-2 seconds for 1M RAM). During this time period the virtual group is unable to schedule any real-time tasks nor is it able to respond to any interrupting devices.

Like the graceful degradation strategy this recovery technique can be managed entirely by the fault masking group which diagnosed itself; it is not necessary to activate the system manager to control the recovery activity. However, other virtual groups communicating with the recovering virtual group must be able to tolerate a 1-2 second dropout. This constitutes an application-specific decision.

5.6.8.2.1.3. Processor Reintegration

In some cases a channel failure will manifest itself as a syndrome error indicating that the processor presented bad data for voting without that channel losing synchronization. In this case the failure could be attributable to a bad RAM location which could be rectified by a memory realignment without the resynchronization of a lost channel or just a glitch in an outgoing message. It is necessary to realign the memory (that is, machine state) of the processor with the other channels in order to correct a recurrent syndrome error in a processor. This strategy has the similar temporal characteristics as processor resynchronization.

5.6.8.2.1.4. Processor Replacement

The intent of the processor replacement strategy is to replace the faulty processor with a spare processor known to be fault free in order to maintain the redundancy level of the virtual group. In this strategy, a spare processor must be located, configured as a member of the redundant virtual group while the faulty processor is configured as a simplex, and the memory of the redundant virtual group be aligned. This option also has the similar temporal characteristics as processor resynchronization.

In this scenario it is necessary that a system-wide task control this process rather than the diagnosed virtual group itself. Because multiple virtual groups are involved in the reconfiguration process these virtual groups must be coordinated globally. The system manager maintains knowledge of the health and configuration of all AFTA components. Consequently, it maintains the information required to optimally decide the updated configuration.

5.6.8.2.1.5. Processor Replacement with Initialization

Because the processor replacement strategy suffers a significant time penalty because of the alignment process, an alternate strategy is posed which closely parallels that strategy. The processor replacement with initialization alternative replaces a faulted processor with a

spare but rather than align the virtual group it initializes all tasks in the virtual group. Task initialization is expected to require significantly less time than the alignment process.

5.6.8.2.1.6. Task Migration

The strategies specified previously have concentrated around the redundancy constraints - either its relaxation (degrade VG) or its maintenance. There may be other constraints to a recovery policy such as maintenance of communication among a redundant virtual group with an I/O device. For example, if a processor of a redundant virtual group is assigned communication over a FCR backplane bus with a specific I/O device and if that processor fails, the I/O task could be transferred to another virtual group with a member in that fault containment region.

This alternative is essentially a single task migration rather than a total transfer of all tasks to a spare processor as would be the case for a memory alignment. The migration of a single active task is very complicated, requiring not only the transfer of the task's stack space but also its global variables which may be scattered throughout memory and, of course, would likely be intermingled with variables of other tasks. Consequently, a task migration could only be a feasible recovery alternative only in circumstances where the migrated task could be transferred and initialized.

5.6.8.2.2. Recovery from Network Element Failure

The failure of a Network Element significantly reduces the reliability of the system. Not only does its failure increase the probability of a system failure because of the loss of this shared resource but also because any processors attached to that failed Network Element are disabled as well despite the health of those processors. If the processors are members of redundant virtual groups, the redundancy of their associated virtual groups is decreased.

Because of the criticality of the Network Elements for Byzantine resilient communications, it is important that recovery of a failed Network Element be attempted. In fact, the recovery of a Network Element is implicit in the design of the network element architecture so that the recovery of *only* the failed NE is much less disruptive of system operations than recovery of a failed processor. This strategy is described as the Network Element resynchronization option. However, because of the nature of the failure, it is not always possible to recover a failed Network Element. For the latter situation, a Network Element masking option is presented.

5.6.8.2.2.1. Network Element Resynchronization

The reintegration of a Network Element is a multiple step process which can include reintegration of the processors into their respective redundant virtual groups:

- 1) The Network Element must be reset to initialize its internal state.
- 2) The Network Element must resynchronize itself with the other Network Elements.
- 3) Processors communicating directly with the resynchronized Network Element can be reintegrated with the other members of their corresponding virtual groups. This could be accomplished using one of the processor recovery strategies described above.

Because the system manager assumes responsibility for the diagnoses of a Network Element, it would also reset the faulted Network Element via a voted reset (See Section 4). This automatically initiates a synchronization methodology within the Network Element which attempts to perform an initial synchronization (ISYNC). When the network fails to detect an initial synchronization from the other Network Elements, it initiates a resynchronization phase in which it assumes that the other Network Elements are synchronized and it itself is desynchronized. When the resynchronization has succeeded, the Network Elements align the configuration tables so that each Network Element has a consistent view of the system configuration. In this system configuration each processor on this failed Network Element has assumed a new status as a simplex. If they had been members of redundant virtual groups, they can be resynchronized and realigned using a processor recovery strategy.

5.6.8.2.2.2. Network Element Masking

In some cases a Network Element may not behave correctly even after repeated attempts to recover that failed Network Element. It may fail to respond to a voted reset or to synchronize with the other Network Elements or it may exhibit faulty behavior shortly after reintegration. In these cases, it is necessary to permanently disable that Network Element via a configuration update masking out the failed Network Element. This message (issued by the system manager) will cause the other Network Elements to disable the faulted network element's data and clock inputs.

5.6.9. Transient Fault Analysis

When a component exhibits faulty behavior, it is important to determine if this failure resulted from a transient condition or from a permanent malfunction. If the failure can be deemed a transient failure then system resources can be utilized most efficiently because the component can be reintegrated into the functioning system. Since transient failures are as-

sumed to be caused by some temporary environmental condition (e.g., a power surge), they are expected to disappear with time. Permanent malfunctions, on the other hand, are caused by breakdowns of the AFTA hardware that must be physically repaired.

A transient fault strategy will be implemented which resets the component and repeats the test suite. A subsequent failure is indicative of a permanent failure and the component would be disabled.

Transient fault analysis can be implemented by one of the following basic strategies:

- 1) A *transient recovery* policy
- 2) A *wait-and-see* policy, or
- 3) A *no transient fault analysis* approach.

5.6.9.1. Transient Recovery Option

The *transient recovery* policy would immediately disable the faulty component and implement a recovery policy to reintegrate the faulty component into the system. After successful integration, if the component did not fail again during a probationary period, it is deemed to have suffered a transient fault. Figure 5-36 depicts the algorithm for transient recovery.

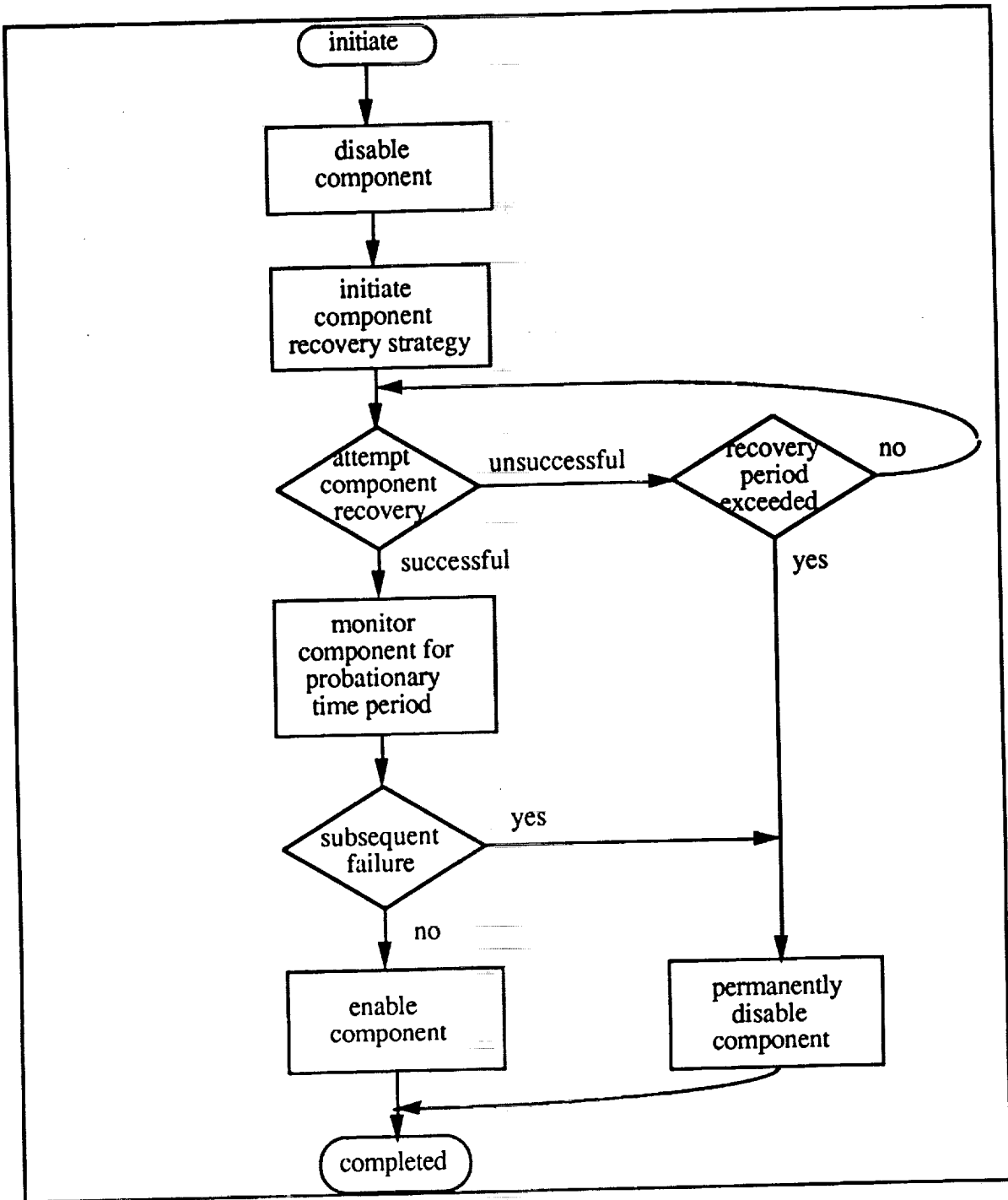


Figure 5-36. Transient Recovery Algorithm

The distinction between transient and hard failures defines the two functions of the transient recovery option:

- It decides when it is appropriate to attempt to component recovery.
- Once a component has been reintegrated, it monitors its health for a brief probation period before declaring it fully recovered.

Using the transient recovery method, a component recovery is periodically attempted. If indeed a component encountered a transient fault, it is desirable to recover the component quickly. Conversely, if the component suffered a permanent failure, it is highly desirable to avoid excessive computing resources to revive this component. Transient recovery balances these two requirements by initially assuming that any particular fault is transient (it has been observed that 50 to 80 percent of all faults in computer systems are transient) and automatically attempting a recovery. As time passes without the component being recovered, it becomes more likely that the fault is a hard failure rather than a transient, and transient analysis makes the recovery attempt less often. After a certain period it can be reasonably assumed that the failure is a hard failure; therefore, the transient analysis function permanently disables the component.

Additionally, it has been noted that permanent failures tend to manifest themselves sporadically. A component may be recovered according to the above criteria, but may immediately fail again. Transient fault analysis attempts to prevent this situation by regarding a recovered component as recovered only on a trial basis. If the component passes its trial period without further errors, it is regarded as fully recovered and can be incorporated into the AFTA configuration. On the other hand, if the component fails during the probationary period, the component is permanently disabled.

If transient recovery fails to reintegrate the faulted component, an alternate recovery strategy can be invoked. This might be the case if a processor fault which initiated the processor reintegration strategy subsequently suffered another fault. It may be appropriate that the processor be permanently eliminated from the virtual group via graceful degradation or processor replacement.

A transient fault can cause a state change which may not disappear with time. Using the transient recovery methodology, the diagnosed component is essentially initialized, reintegrated into the operational system and, after the trial period, is exonerated of being faulty. Because the transient recovery policy attempts to return the component to an operational state, the transient recovery policy is the most ideal option.

5.6.9.1.1. Processor Recovery

When a processor is diagnosed as faulty, a processor can be disabled yet still maintain its identity as a member of a virtual group. These processors can be recovered in two basic ways which depend upon the manifestation of the processor fault using either the processor resynchronization or the processor integration strategy. As indicated previously, both recovery options require a significant amount of time because of the memory alignment process integral to these recovery strategies.

5.6.9.1.2. Network Element Recovery

When a Network Element has exhibited faulty behavior and is immediately disabled, that Network Element is reset causing it to lose synchronization with the other network elements. Since the Network Element communicates with some number of processors which may be members of redundant virtual groups, each of those virtual groups consequently loses a channel. Because the Network Elements are integral to Byzantine resilient communications, it is highly desirable a Network Element which has suffered a transient fault be reintegrated. A recovery policy should include the reintegration of the Network Element and, if possible, recovery of all processors as well.

5.6.9.2. Wait and See Transient Analysis Option

The transient recovery option performs the analysis of a transient failure condition by attempting reintegration of the diagnosed component and analyzing the results. However, any recovery option to reintegrate a failed component is timely and may be in conflict with the mission requirements at the time the recovery is attempted. For these reasons a more conservative approach is posed as a substitute.

The *wait-and-see* transient analysis policy does not disable a component until the fault condition existed for a prescribed period of time. If the fault persists, then the component is judged to have endured a permanent fault. If the fault disappears, then it is assumed that the component suffered a transient fault.

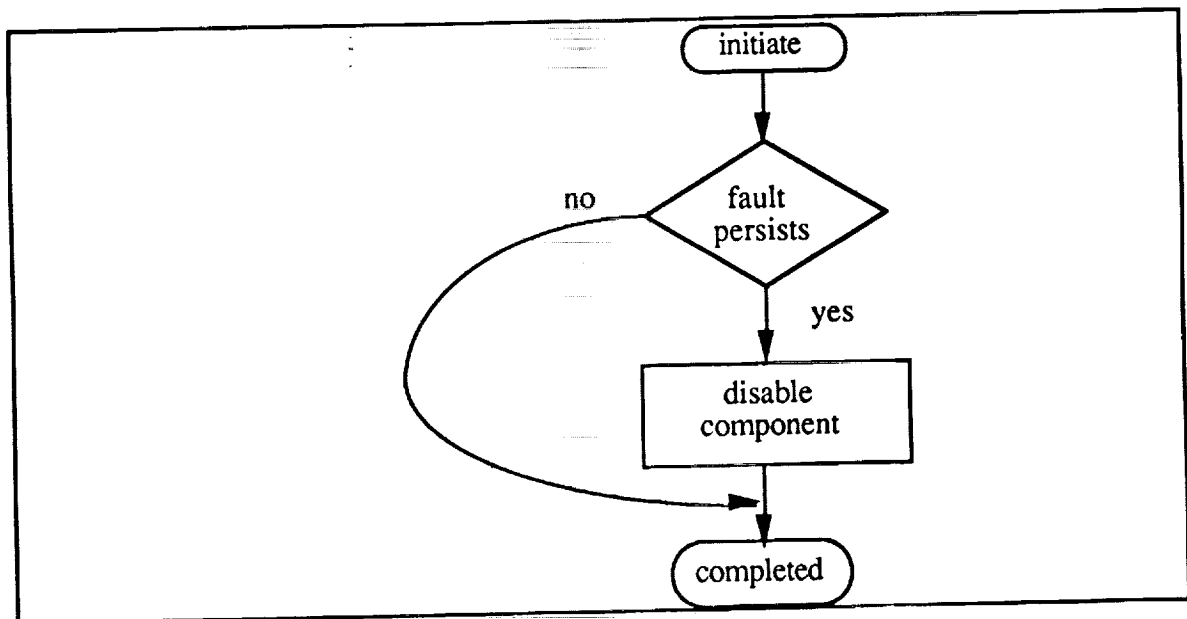


Figure 5-37. Wait and See Transient Fault Analysis algorithm

This policy is particularly attractive when the mission constraints do not permit the implementation of a component recovery strategy and hence, a transient recovery option. This option may be used in conjunction with a graceful degradation strategy for processor failures.

5.6.9.3. No Transient Fault Analysis Option

A *no transient fault analysis* approach may be selected which immediately disables the faulty component and does not attempt to reintegrate the component diagnosed as failed.

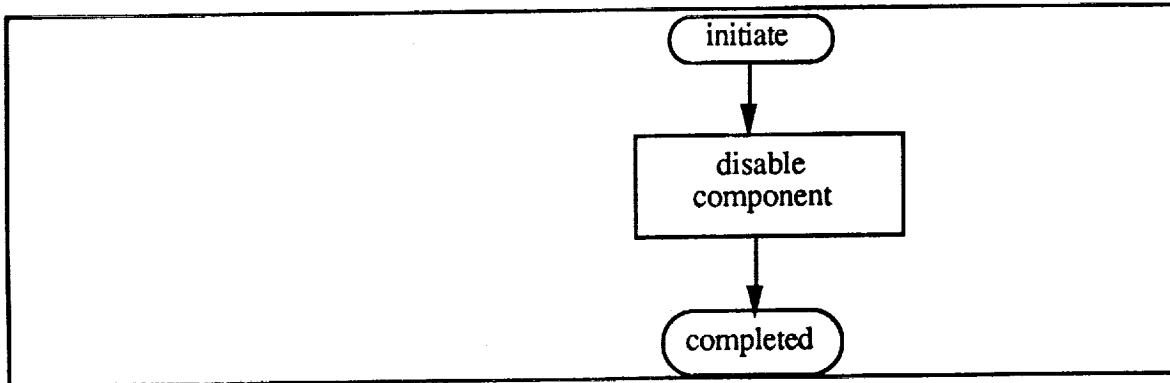


Figure 5-38. No Transient Fault Analysis algorithm

5.6.9.4. Hybrid Transient Fault Analysis Option

When a Network Element fails it is highly desirable to attempt to recover at least the Network Element in order to maintain a system which is resilient to Byzantine failures. However, because disabling the Network Element actually desynchronizes it from the other Network Elements, it is imperative that a failure exist in the Network Element with a high degree of certainty. For these reasons, a hybrid transient analysis is presented which combines the functionality of the transient recovery and the wait-and-see strategies. This is depicted in Figure 5-39.

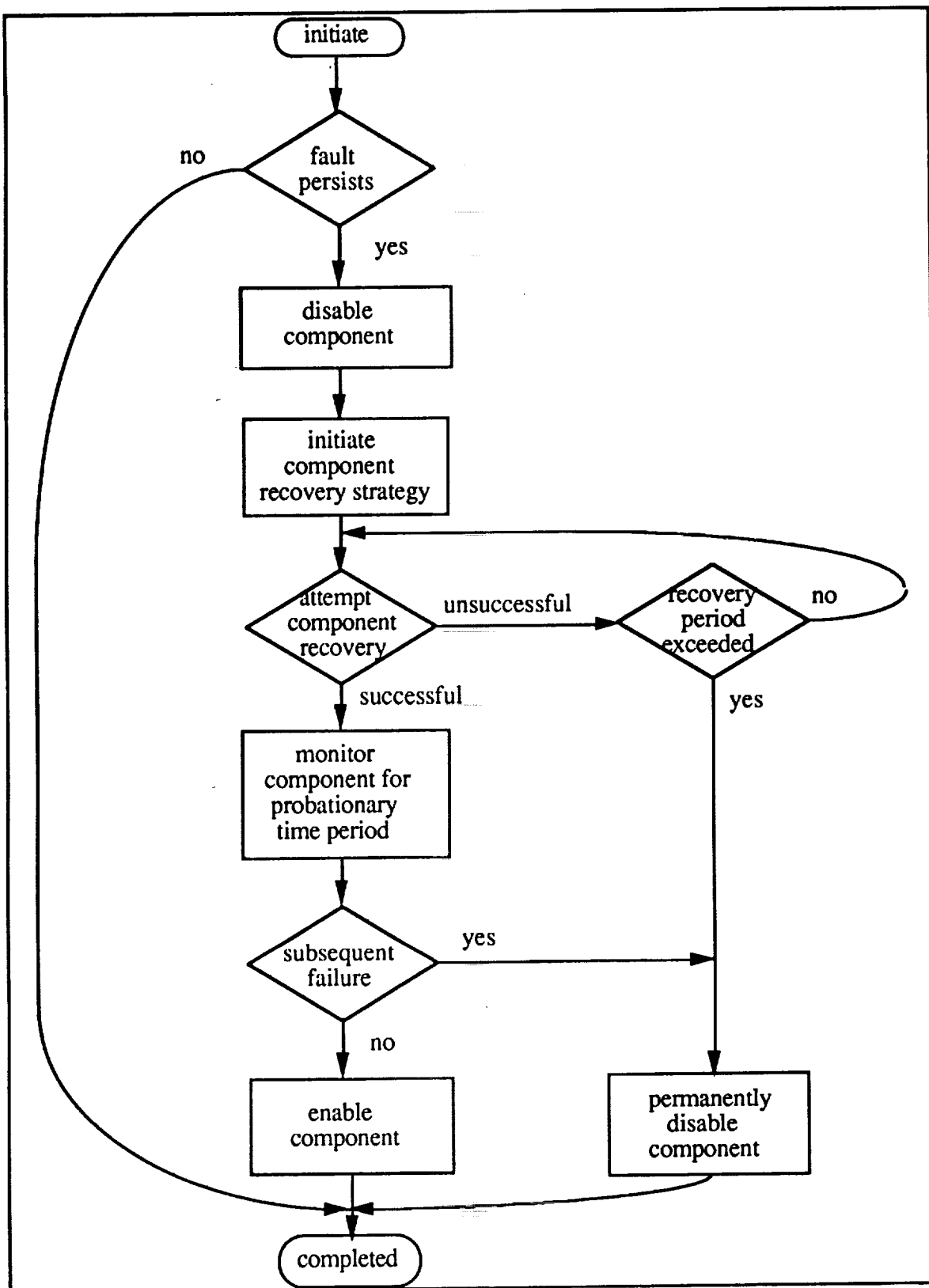


Figure 5-39. Hybrid Transient Fault Analysis algorithm

5.6.9.5. Intermittent Fault Analysis

Typically, a component which is failing exhibits faulty behavior sporadically some time period before its failure becomes unquestionably permanent. If a fault in a component re-occurs within a predefined time threshold then the fault, originally classified as a transient, will be reclassified as an intermittent. This intermittent failure interval is assumed to be greater than the probationary period.

5.6.9.6. Transient Fault Analysis Option and System Modes

It may desirable to maintain multiple recovery options for each component which are a function of the system mode. The system requirements during power-on or standby may be much less stringent than during the operational mode. Figure 5-40 presents a possible correlation of transient fault analysis methodologies for processor and Network Element failures in each system mode.

failed component type \ system mode	power-on	standby	operational
processor failures	transient recovery	transient recovery	wait-and-see transient analysis
network element failures	transient recovery	transient recovery	hybrid transient recovery

Figure 5-40. Possible Mapping of Transient Analysis Options to System Modes

5.6.10. Fault Logging

In order to support automatic fault logging and maintenance recording, each fault containment region will be equipped with a mass memory device. This memory device will contain non-volatile RAM to ensure that neither maintenance records for each component nor faults detected during the operational modes disappear when power is lost.

Information regarding the component identifier, number of power-on cycles, time since power-on, Greenwich Mean Time, fault description, and system configuration will be

maintained in the data log for retrieval at a later time by line maintenance personnel. In addition, this repository will also keep maintenance records such as when each component was installed or serviced and when the M-BIT test suite last exercised each component.

Any processor within a fault containment region may access the mass memory via the FCR backplane bus. During I-BIT and M-BIT non-fault tolerant mode testing all processors may access the mass memory to log results of their self tests. In order to facilitate later fault reporting, the fault information stored during the I-BIT and M-BIT self tests will be disseminated during fault tolerant operations so that all mass memory devices will maintain identical copies of the fault status of each component. Even after the distribution of the fault information, however, it is still possible that this data will be non-congruent especially if the communication medium is disrupted between the diagnosing entity and the mass memory as would be the situation when a Network Element, mass memory device or FCR backplane bus fails.

During fault tolerant operations the diagnosing virtual group will transfer fault information to the system manager which coordinates and distributes this information for storage in all mass memory devices.

Because the fault log maintains information concerning all faults which have occurred irrespective of the test mode or the transient analysis policy active at the time of the fault detection, there may be logged faults which are indicative of a transient fault and, consequently, are not reproducible (that is, cannot duplicate). The M-BIT tests are useful to discern whether or not a logged fault was a transient by extensively exercising all functional components of the LRM identified as faulty.

5.6.11. Fault Reporting

All fault information will be maintained in the mass memory devices in each FCR. These will be the primary repository for all fault information. During non-fault tolerant operations a processor in each FCR will be capable of extracting this information for transmittal to a fault reporting device attached to the FCR backplane bus. During fault tolerant operations a single processor in the fault containment region will be responsible for extracting fault information from its local mass memory and for communicating that information to the system manager via fault tolerant message exchanges. The system manager will format the information for the displays.

Fault status will be reported on three different types of displays – a cockpit display unit (CDU), a portable intelligent maintenance aid (PIMA) and a fault annunciator panel (FAP).

5.6.11.1. Cockpit Display Unit

The CDU is a CRT display with a small screen located in the cockpit for display of system status to the vehicle operator. This display may have three levels of detail with regard to the identification of a faulty component:

- 1) AFTA system status,
- 2) LRU level status or
- 3) LRM level status.

The AFTA system status is merely an indicator representing the GO/NO GO status of the AFTA system. The AFTA status represents the availability of system components to achieve the minimum dispatch complement (MDC) required for the mission critical operation.

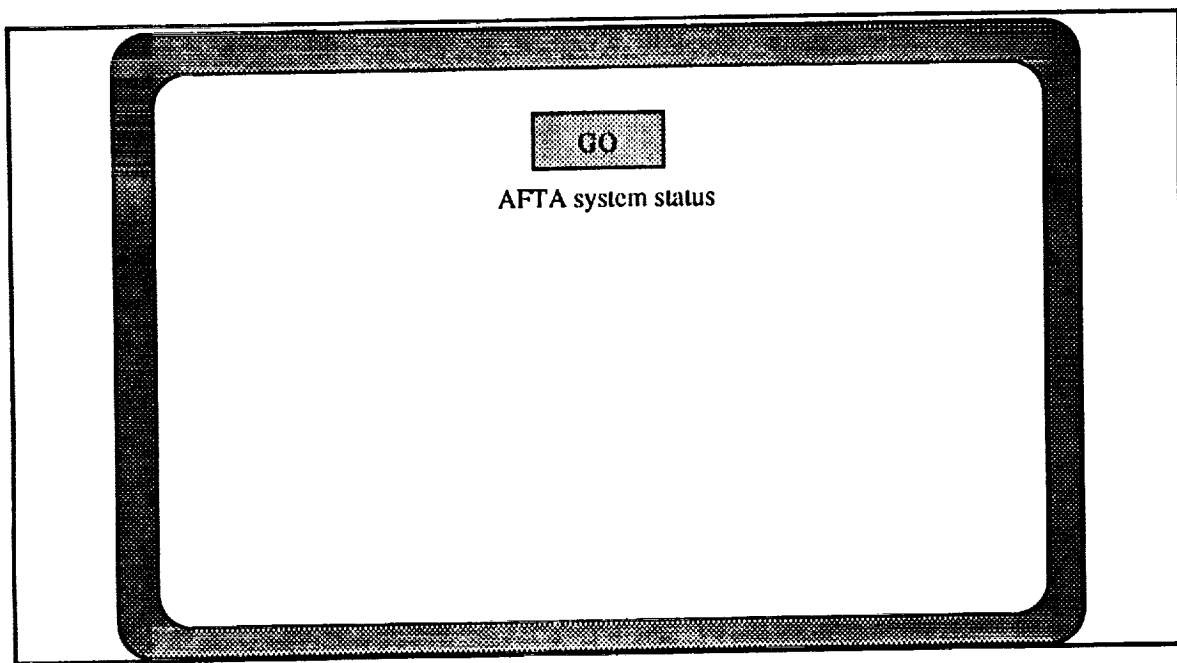


Figure 5-41. AFTA System Level Display

The LRU level displays the status of each fault containment region. This status is essentially dependent upon the availability of shared components within the fault containment region such as the FCR backplane bus or the Network Element. The failure of any of these components renders a NO GO indication. Because the ability of the AFTA to achieve MDC is not wholly dependent upon the availability of the LRU but depends upon the availability of processors and I/O devices as well, the display of the LRU status should also include the display of the AFTA system status.

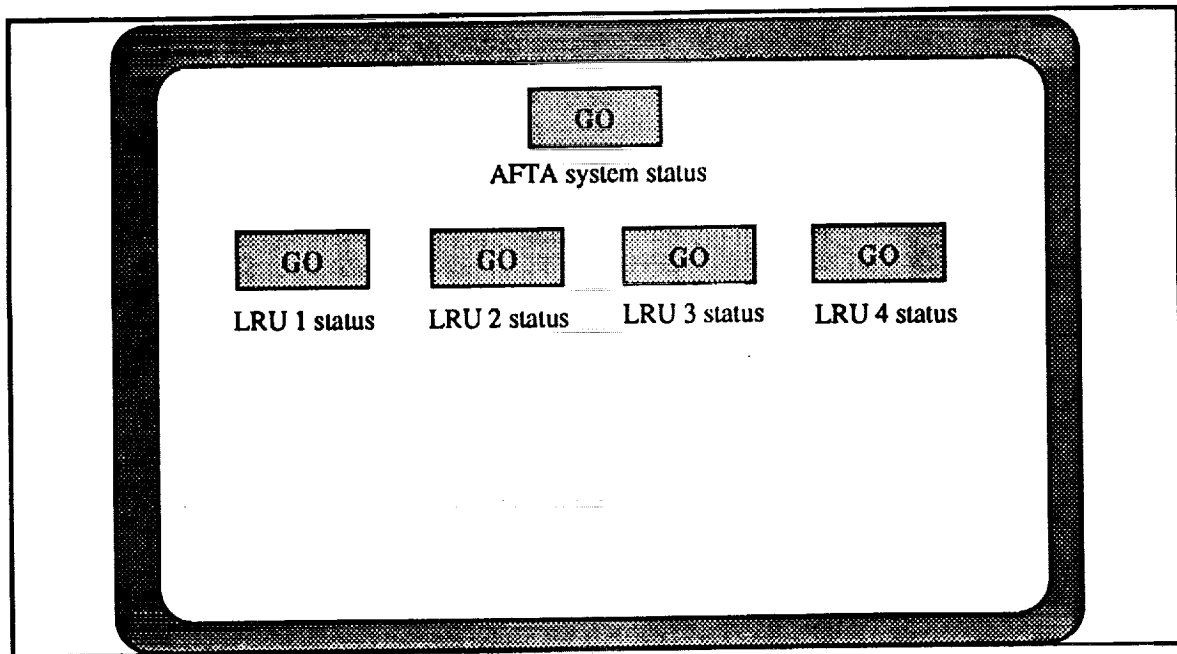


Figure 5-42. LRU Level Display

The LRM level is the most detailed because it represents not only the status of the network element but all other individual components (that is, processors, I/O devices, power conditioner, mass memory device, and FCR backplane bus) as well. This display option can precisely depict the cause of some failures. For instance, if the AFTA system status indicated a NO GO state, the LRM level depicts precisely which component failures generated this state. It may have been caused by the failure of a single crucial component such as a Network Element or it may have resulted by attrition of any combination of I/O device or processor failures. A possible representation for the LRM display is depicted in Figure 5-43.

GO								
AFTA System Status								
LRU 1	GO	GO	GO	GO	GO	GO	GO	NO GO
	PE 1	PE 2	PE 3	PE 4	NE	MM	BUS	PC
LRU 2	GO	GO	GO	GO	GO	GO	GO	GO
	PE 1	PE 2	PE 3	PE 4	NE	MM	BUS	PC
LRU 3	GO	GO	GO	GO	GO	GO	GO	GO
	PE 1	PE 2	PE 3	PE 4	NE	MM	BUS	PC
LRU 4	GO	GO	NO GO	GO	GO	GO	GO	GO
	PE 1	PE 2	PE 3	PE 4	NE	MM	BUS	PC

Figure 5-43. LRM Level Display

The CDU will only be updated while the system is in either a standby or operational mode. Communication with the CDU requires that the AFTA be operating synchronously with fault tolerant message exchanges.

5.6.11.2. Portable Intelligent Maintenance Aid

The PIMA is a unit specifically dedicated to aid in maintenance diagnostics. Ideally, it would resemble a laptop computer with a display, keyboard or buttons, and a printer. It is employed to initiate maintenance diagnostic testing (that is, M-BIT), to interrogate the AFTA for detailed fault information logged during operations for display or printing purposes and to extract maintenance records for each component.

The PIMA is plugged into a socket which is located on the outside of each LRU as well as at other strategic vehicle locations. However, for flexibility in performing maintenance operations, there are essentially two connection options – AFTA system level and LRU level. The distinction in these options revolves around the nature of the status of the AFTA. With the AFTA system level option, a maintenance officer would plug the PIMA into any socket and would be able to extract fault and maintenance information system-wide (that is, from all fault containment regions). Alternatively, with the LRU level option the PIMA would require plugging into the specific LRU from which fault and maintenance information was desired. The former option would require that the AFTA hardware be in standby mode in order to communicate all system information to the selected socket. The option is selectable from the PIMA.

5.6.11.3. Fault Annunciator Panel

The FAP is a panel displaying the "go/no go" status of each component in a fault containment region. The panels are physically located in each LRU in close proximity to the LRMs. It is implemented as a series of mechanical switches which are controlled by the AFTA and which maintain their status when power is turned off. Each switch corresponds to a single LRM in an easily identifiable pattern so that an LRM which is designated faulty component can be readily identified and extracted for maintenance or replacement.

5.7. I/O Services

The Army Fault Tolerant Architecture I/O Services provide efficient and reliable communication between the user and external I/O devices (sensors and actuators). It is logically segmented into two functional modules: the I/O User Interface and the I/O Communication Manager (illustrated in Figure 5-44). Applications engineers use the I/O User Interface to define the required I/O activity during the specifications phase. During the execution phase, the I/O Communication Manager controls the processing of the I/O requests.

The I/O User Interface and I/O Communication Manager are dependent processes, as depicted in Figure 5-45. The Interface interacts with the application tasks to create an I/O request database. Further, the Interface and Communication Manager exchange control and status information; the output data and control commands are destined for the I/O devices while the input and status data are sent to the application tasks. Additionally, the Communication Manager retrieves information from the I/O request and I/O device databases and interchanges data with the I/O devices.

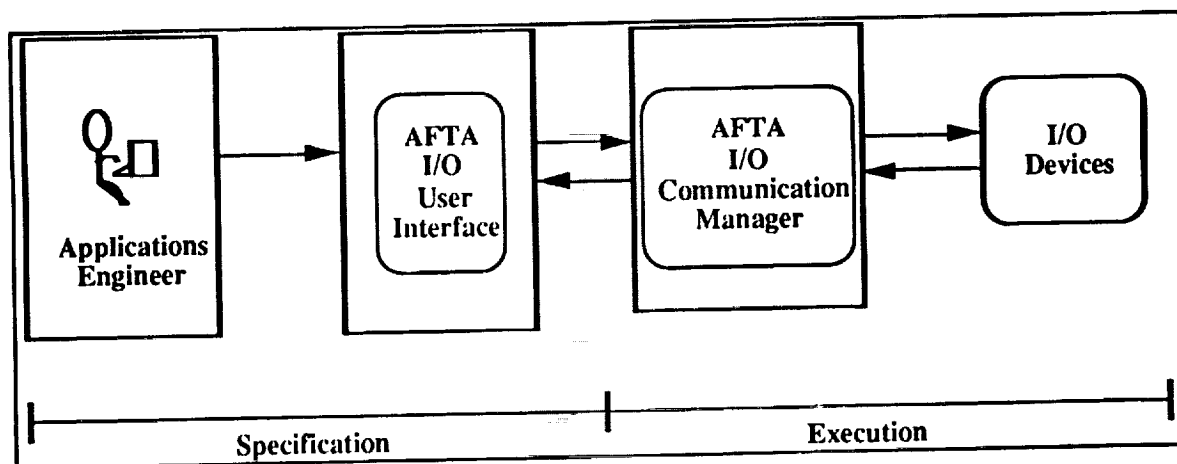


Figure 5-44. The AFTA I/O Services

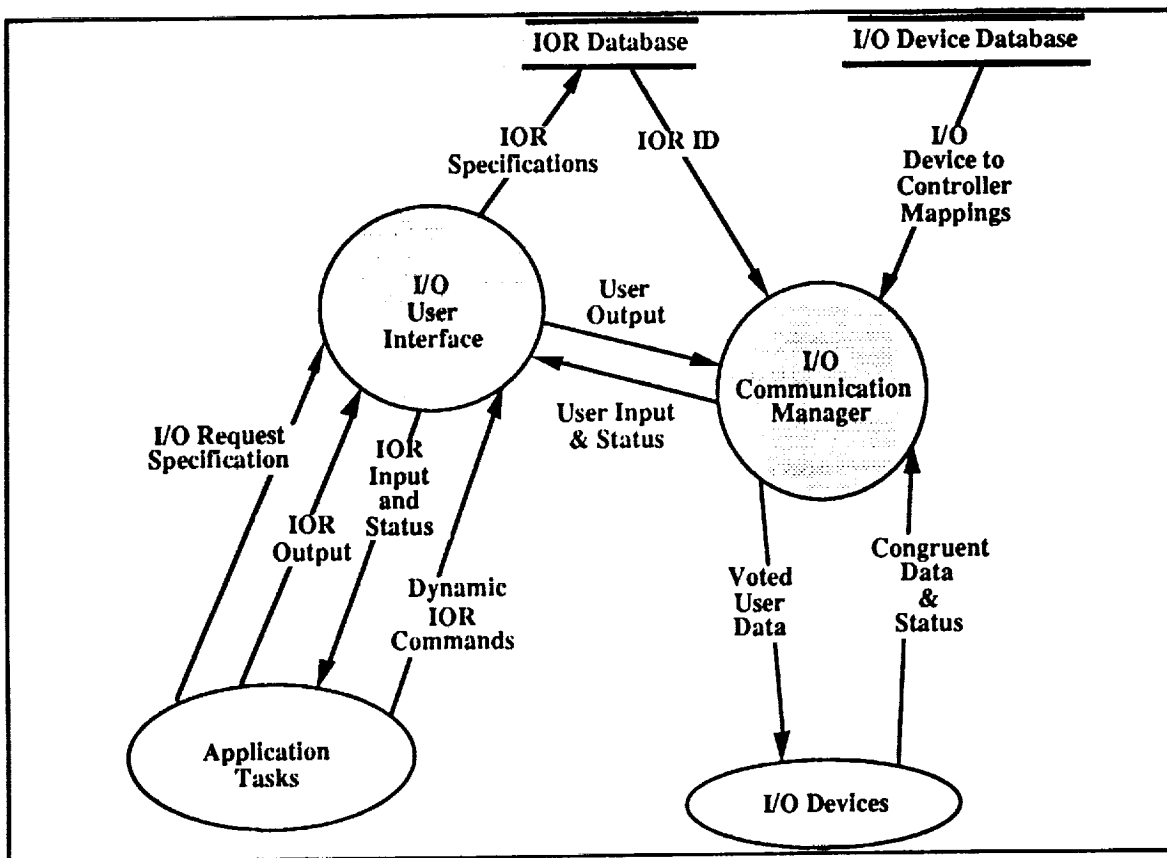


Figure 5-45. The I/O User Interface and I/O Communication Manager

The preliminary design of the AFTA I/O User Interface is detailed in Section 5.7.1 while the I/O Communication Manager is described in Section 5.7.2. Sections 5.7.3 and 5.7.4 use examples to clarify the description of the AFTA I/O Services: Section 5.7.3 highlights the dependence between I/O requests' execution and processing times.

5.7.1. The AFTA I/O User Interface

The discussion of the I/O User Interface is separated into three sections: I/O User View, I/O Request Construction, and I/O Data Access.

5.7.2. Input/Output User View

There are three desired characteristics of the AFTA I/O process. First, the load modules of different members of a redundant VG must be identical, even if only a subset of the members actually execute the I/O operation. The second requirement is that the control flows of redundant VGs executing I/O must be similar if not identical, even if only a subset of the members actually execute the I/O operation; heterogeneous I/O must not be allowed to induce sufficient skew to force the desynchronization of a redundant VG. Finally, when

redundant I/O is accessed, it is important that the copies of the I/O device be accessed at very close to the same time.

In addition, it is currently planned that all I/O activity will be synchronized with frame boundaries. That is, even though I/O requests may be completed at any time within the minor frames, their data will only be exchanged at the beginning and/or end of the frames.

5.7.3. I/O Request Construction

The AFTA I/O User Interface is a flexible framework which is easily tailorable to meet the reliability and performance requirements of avionics applications that access external devices. The Interface is easy to use; the applications designer can specify the I/O activity in a straightforward manner. In addition, the Interface provides all the tools necessary to meet AFTA's I/O needs.

The AFTA I/O Services can either communicate directly or indirectly with I/O devices (sensors and actuators). Direct communication is achieved by sending data and command information immediately to the device. Indirect communication utilizes an I/O controller to access a device. This intervening mechanism accepts data and control commands from the VG and then manages the I/O operation.

The AFTA I/O Services support two general types of I/O activity: sequential and concurrent. *Sequential I/O* requires that the VG completely supervise the activity; that is, it must block itself until the I/O operation has finished. Accordingly, the VG and the I/O devices are tightly synchronized during the I/O activity. This is necessary to communicate with I/O controllers or devices that have limited processing capabilities such as A/D converters or "dumb terminals".

Alternatively, *concurrent I/O* allows the VG to perform other tasks while the I/O is being processed. The VG downloads data to the controller, sends an "start" command, and then executes another process. After the I/O has completed, the VG collects the resultant input data. The concurrent I/O capability is provided to maximize AFTA's processing throughput. To permit this parallel I/O - VG processing, smart hardware such as an Ethernet or 1553 controller is necessary.

The applications engineer defines the required I/O activity. This is accomplished by specifying one or more I/O requests. The I/O specifications are constructed in an hierarchical manner, beginning with transactions, continuing with chains, and ending with the I/O requests. Figure 5-46 depicts these components and how they are related.

5.7.4. I/O Transactions

A transaction is an autonomous command (or command and response) sequence, permitting interaction between a VG and an I/O device (IOD). In general, an application can create three types of transactions:

- An *input* transaction which is a sequence of instructions that waits for information from an IOD.
- An *output* transaction which consists of a sequence of instructions that sends information to an IOD and does not expect a response.
- An *input/output* transaction which involves a sequence of instructions that requests information from an IOD followed by a sequence of instructions that waits for the IOD response.

If the transaction directly accesses an I/O device or controller that has limited processing capability, then the VG must perform a sequential I/O operation. In contrast, if the device or controller is relatively intelligent, then the VG can execute a concurrent I/O transaction.

The parameters necessary to specify input and output transactions and examples of their initialization are illustrated in Figures 5-47, 5-48, and 5-49 respectively. These fields are defined as follows:

- *Transaction_Type*. This parameter indicates whether the transaction is an input, output, or input/output operation.

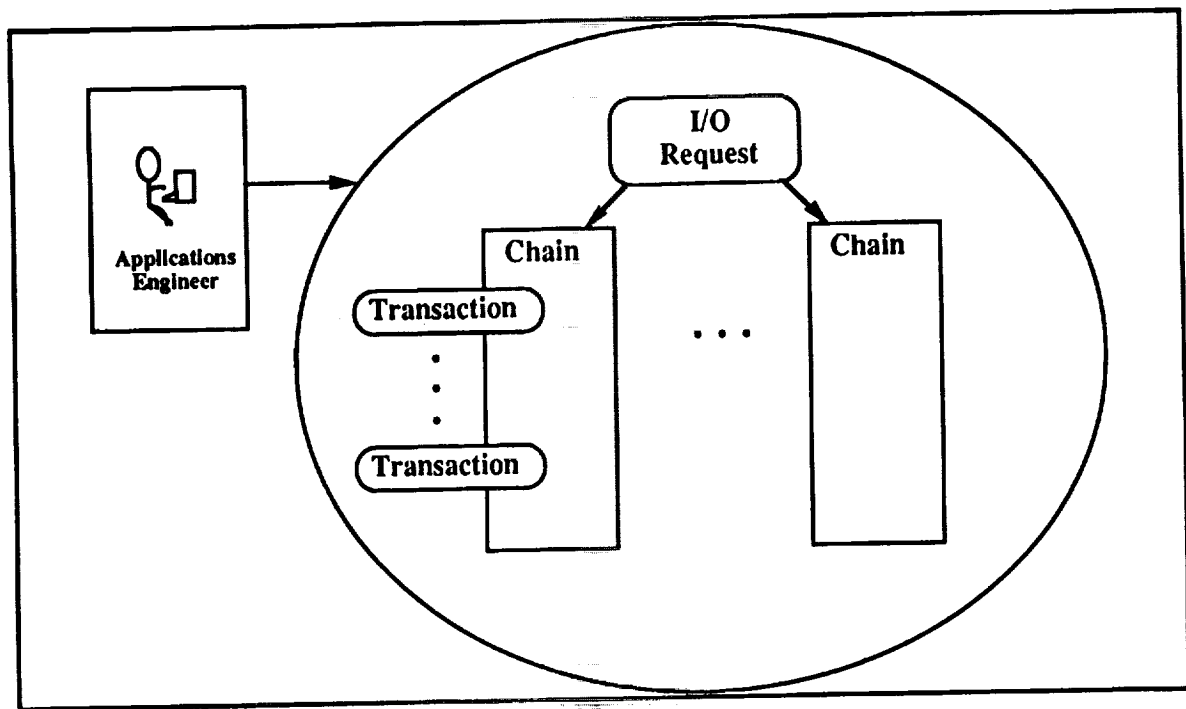


Figure 5-46. I/O Transactions, I/O Chains, and I/O Requests

- ***IOD_Identifier***. This is the I/O device identifier. It is used to indicate the corresponding device driver and determine the device's address.
- ***Num_Input_Bytes***. The number of input data bytes that are expected by the transaction.
- ***Num_Output_Bytes***. This parameter specifies the number of output data bytes that will be transmitted by the transaction.
- ***Dynamic_or_Static***. This field indicates whether the output data for this transaction is dynamic (changes with time) or static (time-invariant).
- ***Time_Out***. This is the worst case time that is waited by the VG or an I/O controller for the arrival of an incoming data byte (in microseconds).
- ***Input_Buffer***. Address of the buffer on the VG in which the input data will be stored.
- ***Output_Buffer***. Address of the buffer on the VG from which the output data will be transmitted.
- ***SC_Transactions***. An array of transaction identifiers that depicts the transactions that are involved in this transaction's I/O source congruency (SC) algorithm.
- ***SC_Source***. This parameter identifies the source congruency algorithm that will be used. The options are: From_A, From_B, From_C, From_D, and Voted.

```

INPUT_TRANSACTION_INFORMATION :=
( TRANSACTION_TYPE      => INPUT;
  IOD_IDENTIFIER        => GUIDANCE_COMMAND;
  NUM_INPUT_BYTES       => 12;
  TIME_OUT              => 128;
  INPUT_BUFFER          => BUFFER'ADDRESS;
  SC_TRANSACTIONS       => SC_TRANS_ARRAY;
  SC_SOURCE              => FROM_A);

```

Figure 5-47. An Input Transaction Record

```

OUTPUT_TRANSACTION_INFORMATION :=
( TRANSACTION_TYPE      => OUTPUT;
  IOD_IDENTIFIER        => ENGINE_ACTUATOR;
  NUM_OUTPUT_BYTES      => 6;
  DYNAMIC_OR_STATIC     => DYNAMIC;
  OUTPUT_BUFFER         => BUFFER'ADDRESS);

```

Figure 5-48. An Output Transaction Record

```

INPUT_OUTPUT_TRANSACTION_INFORMATION :=
( TRANSACTION_TYPE      => INPUT_OUTPUT;
  IOD_IDENTIFIER        => ALTITUDE_SENSOR;
  NUM_INPUT_BYTES       => 16;
  NUM_OUTPUT_BYTES      => 2;
  DYNAMIC_OR_STATIC     => STATIC;
  TIME_OUT              => 255;
  INPUT_BUFFER          => IN_BUFFER'ADDRESS;
  OUTPUT_BUFFER         => OUT_BUFFER'ADDRESS;
  SC_TRANSACTIONS       => SC_TRANS_ARRAY;
  SC_SOURCE              => VOTED);

```

Figure 5-49. An Input/Output Transaction Record

```
CREATE_TRANSACTION ( TRANSACTION_ID,  
TRANSACTION_INFORMATION );
```

Figure 5-50. The Create_Transaction Procedure

After the transaction's parameters have been defined, the Create_Transaction procedure (shown in Figure 5-50) must be invoked to inform the AFTA I/O Services of the transaction's existence. This call requires two fields:

- *Transaction_ID*. An identifier that is returned to the application by the Interface, allowing the transaction to be easily referenced in the future.
- *Transaction_Information*. The record, described above, which depicts the values of the transaction's fields.

5.7.5. I/O Chains

A chain is a set of transactions, grouped to efficiently utilize the communications bandwidth. Chains are executed as autonomous units. The transactions of a chain are executed serially, and all are either sequential or concurrent I/O. If the chain is a concurrent I/O operation, then all transactions in the chain must utilize the same I/O controller.

```
CREATE_CHAIN( CHAIN_ID,  
CHAIN_I/O_TYPE => CONCURRENT;  
TRANSACTION_ARRAY => TRANSACTION_ID_ARRAY);
```

Figure 5-51. The Create_Chain Procedure

After a chain's transactions have been specified, the Create_Chain procedure, depicted in Figure 5-51, should be executed to link the chain to its transactions. It has three parameters:

- *Chain_ID*. An identifier that is returned to the application allowing the chain to be easily referenced.
- *Chain_I/O_Type*. A field that indicates whether the I/O is sequential or concurrent.
- *Transaction_Array*. The array of transactions that comprises the chain.

5.7.6. I/O Requests

An I/O request (IOR) is one or more chains of transactions. All I/O is structured as I/O requests.

Each chain of an I/O request is executed nearly simultaneously. This requirement is desirable because it allows two or more chains to request redundant information. If all chains are executed within a negligible skew, then data from redundant sensors can be retrieved at nearly the same time and the information can be subsequently compared to mask invalid data.

```
CREATE_I/O_REQUEST( IOR_ID,
                    RATE_OF_IOR      => R2;
                    EXECUTION_FRAMES => (0, 4);
                    CHAIN_ARRAY      => CHAIN_ID_ARRAY;
                    EXECUTION_TIME   => IOR_EXEC_TIME);
```

Figure 5-52. The Create_I/O Request Procedure

After an I/O request's chains have been specified, the Create_I/O Request procedure, illustrated in Figure 5-52, should be invoked. This call has five parameters:

- *IOR_ID*. An identifier that is returned to the application allowing the I/O request to be easily referenced.
- *Rate_of_IOR*. A parameter that indicates the rate group of the I/O request, either RG4 (100 Hz.), RG3 (50 Hz.), RG2 (25 Hz.), or RG1 (12.5 Hz.).
- *Execution_Frames*. The frames in which this I/O request will be executed (started in the beginning of the frame). It will be some combination of numbers ranging from (-1) through 7. If the field is set equal to (-1), then the I/O request is started in the beginning of the frames in which it is processed. Strict control over the execution and processing frames is desirable to efficiently manage combinations of I/O requests that stress the 10 ms. throughput bounds.
- *Chain_Array*. The suite of chains that comprises the I/O request.

- *Execution_Time.* The worst case execution time for the I/O request. It is equal to the I/O request time out; that is, the total amount of time the VG waits before presuming that the I/O request has completed execution.

5.7.7. I/O Data Access

The applications user perceives all I/O devices to be memory mapped even though many devices may be connected to the VG through I/O controllers. The appearance of memory mapped I/O is attained by establishing data sections in VG memory to emulate the data regions in the I/O devices or controllers. These memory buffers are allocated by the application and their addresses are passed to the I/O Services when the transactions are defined. The applications engineer communicates to I/O devices by simply writing to and reading from the corresponding local memory regions.

The reading/writing protocols between the VG and the I/O device are transparent to the user. When the I/O request is executed, the I/O Communication Manager reads data from the request's output buffers and sends it, along with command information, to the destination I/O devices. After the I/O request has been processed, the data returned by the devices is stored into the application's input buffers.

Since the application tasks and the I/O Communication Manager share I/O data buffers, both processes could simultaneously update the same memory area. For example, the following scenario could occur: (1) an application task begins to modify a data buffer; (2) it is preempted by the I/O Communication Manager before it completes the update; and (3) the I/O Communication Manager begins to read or write into the buffer. Such simultaneous modification of an I/O request buffer could cause inconsistent or corrupted data to be sent to the I/O devices or read by the application. To prevent (or detect) this condition, the I/O Interface provides procedures that allow the application to lock and unlock the data buffers.

In addition to input data, the I/O Communication Manager receives status information from the I/O devices. This status is recorded in the VG's memory after the I/O request has been executed and processed. (Buffers are allocated by the I/O Interface to store the I/O status when each request is created.) The I/O User Interface provides routines to allow the application to read this data.

The buffer control and status retrieval procedures are discussed in more detail in Sections 5.7.1.3.1 and 5.7.1.3.2, respectively.

5.7.8. The Buffer Control Procedures

To ensure that data buffers for an I/O request are accessed mutually exclusively, the I/O Interface allows the user to lock the request's data regions. Prior to reading or writing

data, the application must invoke the *Lock_I/O_Request_Buffer* procedure (shown in Figure 5-53) to reserve the memory.

```
LOCK_I/O_REQUEST_BUFFER ( IOR_ID,  
                           IN_USE);
```

Figure 5-53. The *Lock_I/O_Request_Buffer* Procedure

The procedure requires two parameters, one provided by the user and the other returned by the Interface. The *IOR_ID* field is specified by the application to identify the request. The *In_Use* field is a boolean returned by the procedure indicating whether or not the request's buffers are currently being accessed by the Communication Manager.

If an I/O request's buffers are not "in use" when *Lock_I/O_Request_Buffer* is executed, then they are reserved by the procedure. After the application data has been read or written, the buffer must be unlocked. This is done by calling the *Unlock_I/O_Request_Buffer* procedure. This routine frees the buffers and thus allows the I/O Communication Manager or another application task to modify the memory region. The procedure, which is illustrated in Figure 5-54, requires that the I/O request ID be provided by the user to identify the buffer.

```
UNLOCK_I/O_REQUEST_BUFFER ( IOR_ID);
```

Figure 5-54. The *Lock_I/O_Request_Buffer* Procedure

If the application tasks and I/O Communication Manager contend for the one or more buffers (i.e., *In_Use* is true), then a fatal scheduling error has occurred. If a collision occurs, the user has to redesign the application tasks, the I/O requests, or both, because contention recovery mechanisms are not provided by the I/O Services.

As discussed, the *Lock_I/O_Request_Buffer* procedure informs user of one of two possible contention scenarios: the I/O Communication Manager is using an I/O request's buffers when an application task wants to access them. The second scenario is the reverse case: the I/O Communication Manager wants to read or store information but an application task has locked the buffers. This is also a fatal scheduling error of which the user must be informed. If this type of fault occurs, an exception, *I/O_Buffer_Contention*, is raised by the I/O Services and passed to the application. To recognize the error, the user must write

an exception handler in each application task. As with *In_Use*, *I/O_Buffer_Contention* requires that either the application tasks or the I/O requests have to be modified. (Figure 5-55 depicts the declaration of the exception and an example exception handler.)

Exception Declaration:

I/O_BUFFER_CONTENTION: EXCEPTION;

Exception Handler:

*WHEN I/O_BUFFER_CONTENTION => WRITE ERROR STATUS;
STOP PROGRAM EXECUTION;*

Figure 5-55. The *I/O_Buffer_Contention* Exception and Error Handler

As mentioned earlier, after an application's I/O requests have been executed, the input data from the I/O devices is deposited into the application's input buffers by the I/O Communication Manager. The data, however, is not necessarily error-free. The user must access the I/O request status information, which is also recorded by the Communication Manager, to determine if any of the data is faulty.

The I/O User Interface allows the application tasks to retrieve status information on an I/O request, chain, or transaction basis. The procedures that enable this access are illustrated in Figure 5-56.

*CHECK_I/O_REQUEST_FOR_ERRORS (IOR_ID,
IOR_HAD_ERROR);*

*CHECK_CHAIN_FOR_ERRORS(CHAIN_ID,
CHAIN_HAD_ERROR,
ALL_TRANSACTIONS_ARE_BAD,
CHAIN_DID_NOT_COMPLETE);*

*CHECK_TRANSACTION_FOR_ERRORS(TRANSACTION_ID,
TRANSACTION_HAD_ERROR,
I/O_DEVICE_STATUS);*

Figure 5-56. The I/O User Interface Status Retrieval Procedures

The *Check_I/O_Request_For_Errors* procedure indicates whether or not any of the I/O request's chains encountered an error during their execution. The *IOR_Had_Error* boolean is set to true if an error has occurred.

The *Check_Chain_For_Errors* procedure designates that a portion of the chain's data is faulty; it sets the *Chain_Had_Error* field to inform the application. It also returns other status information: (1) *All_Transactions_Are_Bad*, a flag that indicates whether or not all transactions in this chain have errors; and (2) *Chain_Did_Not_Complete*, a boolean which informs the user that, for some reason, some of the chain's transactions were not executed.

The *Check_Transaction_For_Errors* procedure returns the *Transaction_Had_Error* parameter to inform the user whether or not a transaction has an error. In addition, it provides the *I/O_Device_Status* field to indicate the status of the I/O device when the transaction was executed (either active or failed).

5.7.9. The AFTA I/O Communication Manager

The AFTA I/O Communication Manager supervises the execution and processing of the I/O requests. It involves two key components: the Nonpreemptable I/O Dispatcher and the I/O Request Tasks. These processes are illustrated in Figure 5-57.

The Nonpreemptable I/O Dispatcher manages the execution of the I/O requests whereas the I/O Request Tasks perform the error detection processing and return the data and status information to the application tasks. These processes are discussed in Sections 5.7.9.1 and 5.7.9.2, respectively. In addition, the scheduling of the I/O Dispatcher and the Request Tasks is outlined in Section 5.7.9.3.

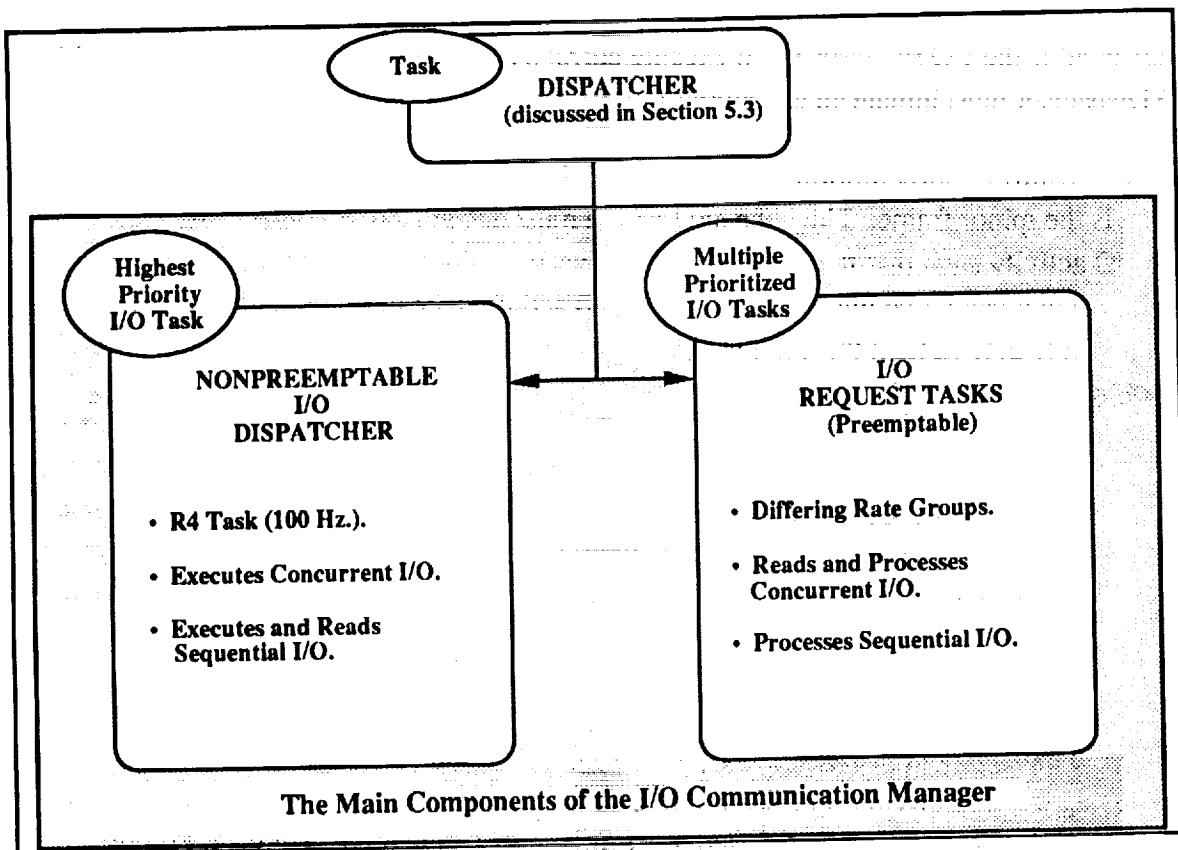


Figure 5-57. The I/O Communication Manager

5.7.9.1. *The Nonpreemptable I/O Dispatcher*

The Nonpreemptable I/O Dispatcher is a task on the VG that manages the execution of the I/O instructions that cannot be interrupted. For the AFTA, two types of nonpreemptable instruction sequences exist: (1) the execution and reading of sequential I/O; and (2) the execution of concurrent I/O.

Sequential I/O must be carefully controlled by the VG, because the associated destination I/O devices have limited processing and storage abilities. Furthermore, applications that utilize sequential I/O often require that data be sent or received quickly and in autonomous batches. If the VG is interrupted, then the I/O operation could be delayed considerably. Thus, the execution of sequential I/O can not be preempted. Additionally, since these I/O devices have minimal memory capabilities, the input and status data for each transaction must be read before a subsequent transaction can be executed. Therefore, the reading of sequential I/O also cannot be interrupted.

In contrast to sequential I/O, concurrent I/O is managed by an intelligent I/O controller, permitting the IOC and VG to run in parallel. The VG, however, must initiate the I/O activity by sending a sequence of "start" instructions to the IOC. This sequence can not be

interrupted if the I/O requests are to execute correctly. Accordingly, the Nonpreemptable I/O Dispatcher must initiate all concurrent I/O.

To ensure nonpreemption, the I/O Dispatcher must complete in less than 10 ms., which is the minor frame. Thus, the application must design and organize its nonpreemptable I/O activity such that the I/O Dispatcher does not exceed this constraint. In addition, the Dispatcher cannot be interrupted by other I/O activity (because it would be delayed and then possibly preempted); thus, it must have the highest priority of the I/O tasks.

The control flow of the Nonpreemptable I/O Dispatcher Task is illustrated in Figure 5-58. The task is scheduled by the Rate Group Dispatcher every 10 ms. Since the type and amount of I/O activity typically varies with each frame, the minor frame number must be determined every time the task is executed. Frame tracking is accomplished by maintaining a modulo 8 counter.

Once the frame number is identified, the Nonpreemptable I/O Task executes the associated I/O requests. The concurrent I/O is executed before the sequential I/O. This allows the VG to execute and process the sequential I/O while the associated I/O controllers are processing the concurrent requests. Some I/O requests may be comprised of both sequential and concurrent I/O chains (referred to as "mixed I/O requests"). They are executed by the Nonpreemptable I/O Task after the concurrent I/O requests but before the sequential I/O requests. This allows the mixed I/O chains to be executed nearly simultaneously while not blocking the execution of the concurrent I/O requests. For clarity, the execution of mixed I/O requests is not explicitly shown in Figure 5-58.

```

task NONPREEMPTABLE_I/O_DISPATCHER is
  FRAME_COUNTER : Integer := 0;
begin
  loop
    -- Wait for the I/O Dispatcher to be scheduled by the Rate Group
    -- Dispatcher.
    WAIT_FOR_SCHEDULE;
    case FRAME_COUNTER is
      0 =>
        for I/O_cnt in Concurrent_I/O_Frame_0
          loop
            Execute_Concurrent_I/O (I/O_cnt);
          end loop;
        for I/O_cnt in Sequential_I/O_Frame_0
          loop
            Execute_Sequential_I/O (I/O_cnt);
          end loop;
        FRAME_COUNTER := 1;

      1 =>
        for I/O_cnt in Concurrent_I/O_Frame_1
          loop
            Execute_Concurrent_I/O(I/O_cnt);
          end loop;

        for I/O_cnt in Sequential_I/O_Frame_1
          loop
            Execute_Sequential_I/O (I/O_cnt);
          end loop;
        FRAME_COUNTER := 2;

      .
      .
      .

      7 =>
        for I/O_cnt in Concurrent_I/O_Frame_7
          loop
            Execute_Concurrent_I/O(I/O_cnt);
          end loop;

        for I/O_cnt in Sequential_I/O_Frame_7
          loop
            Execute_Sequential_I/O (I/O_cnt);
          end loop;
        FRAME_COUNTER := 0;

    end case;
  end loop;
end NONPREEMPTABLE_I/O_DISPATCHER;

```

Figure 5-58. The Nonpreemptable I/O Dispatcher

5.7.9.1.1. The I/O Request Tasks

I/O Request Tasks are primarily responsible for the two functions not completed by the Nonpreemptable I/O Dispatcher: (1) processing sequential I/O; and (2) reading the concurrent I/O data and processing concurrent requests. Since these operations can be time-consuming and preempted by higher priority tasks, they are not performed by the I/O Dispatcher. (As mentioned earlier, the Dispatcher must complete its execution in less than 10 ms.)

To process sequential I/O, the I/O Request Task executes the error detection routines and returns the input data and status information to the application; the redundancy management functions are not performed because the sequential input data has been previously read and distributed to the VG by the Nonpreemptable I/O Dispatcher. In contrast, the I/O Dispatcher does not read the concurrent input data; therefore, the concurrent I/O processing invokes the redundancy management routines as well as executes error detection procedures and stores the input data and status information.

An I/O Request Task is spawned for each I/O request that is created by the user. The Tasks are scheduled by the Rate Group Dispatcher and each falls into one of four rate groups:

- RG4 - 100 Hz. tasks.
- RG3 - 50 Hz. tasks.
- RG2 - 25 Hz. tasks.
- RG1 - 12.5 Hz. tasks.

As described in Section 5.3.1, the RG4 group has the highest priority of the preemptable I/O; RG1 has the lowest. Low priority I/O processes are interrupted by higher priority tasks at the 10 ms. minor frame boundaries. These lower priority tasks are resumed after all higher priority processes have completed.

5.7.9.1.2. Dispatching

The Rate Group Dispatcher uses events to trigger the execution the Nonpreemptable I/O Dispatcher and the Preemptable I/O tasks as well as the other system and application tasks. The Nonpreemptable I/O Dispatcher is the first I/O task to be scheduled in each frame to ensure that it is not interrupted. It is, however, scheduled second overall. The Fault Detection, Identification and Reconfiguration (FDIR) task is executed before the Nonpreemptable I/O Task, because the configuration of the AFTA is important to the I/O Services and FDIR's processing time is short and deterministic.

The execution order of the I/O Request Tasks depends on the request's rate group and inter-group precedence. As discussed in Sections 5.7.2.2 and 5.3.1, RG4 groups are executed before the RG3, RG2, and RG1 groups; RG3 prior to RG2 and RG1; etc. Moreover, within each rate group, precedences determine the scheduling order; that is, tasks with higher precedences execute before those with lower ones.

The AFTA Dispatcher and rate group scheduling paradigm are discussed in detail in Sections 5.2 and 5.3.

5.7.9.2. AFTA Input Output Services: Examples

Two examples are presented to illustrate the interdependence between an I/O request's execution time and the frame in which the I/O request processing is performed. In each example, four I/O requests have been created; one per rate group. For the illustration, a set of parameters was assigned for each request. The *Execution_Frames* and *Chain_Array* parameters were selected arbitrarily and do not effect the example. On the other hand, the *Execution_Time* parameter, which specifies the amount of time necessary for the I/O request to be executed and processed, was chosen carefully and greatly effects the scheduling of the I/O requests. This field is varied to illustrate the I/O requests' execution dependence.

5.7.9.2.1. Example #1: All I/O Requests can be Completed in 10 ms.

In Example #1, all I/O tasks are begun and completed within one frame (FDIR is neglected to clarify the example).

- Frames 0 - 7: The Nonpreemptable I/O Dispatcher and the RG4 I/O request are executed and processed.
- Frames 1, 3, 5, 7: The RG3 request is executed and processed after the I/O Dispatcher and RG4 tasks have completed.
- Frames 3, 7: The RG2 I/O request is executed and processed after the I/O Dispatcher, RG4, and RG3 tasks have completed.
- Frame 7: The RG1 request is executed and processed after all other tasks have completed.

This example, which is specified in Figure 5-59 and depicted in Figure 5-60, comprises the baseline illustration of I/O scheduling in the rate group paradigm.

5.7.9.2.2. Example #2: All I/O Requests can not be Completed in 10 ms.

In Example #2, the *Execution_Time* parameters, which are presented in Figure 5-61, are larger than in Example #1.

- I/O Request #1: *Execution_Time* equals 4.0 ms.; it was 1.3 ms.

- I/O Request #2: *Execution_Time* equals 3.2 ms.; it was 2.2 ms.
- I/O Request #3: *Execution_Time* equals 3.7 ms.; it was 1.9 ms.
- I/O Request #4: *Execution_Time* equals 3.6 ms.; it was 2.1 ms.

The longer execution times cause I/O request #1's execution/processing to be delayed and I/O request #2 to be interrupted. Specifically,

- I/O Request #1 (RG1): This request was supposed to execute in frame #7. However, it was executed and processed in frame #0, because the processing requirements for the I/O Dispatcher, RG4, RG3, and RG2 tasks consumed all of frame #7.
- I/O Request #2 (RG2): This request was started in frame #3 but did not finish. As a result, it was preempted in the beginning of frame #4 by the higher priority tasks, the Nonpreemptable I/O Dispatcher and RG4. After these processes completed, I/O request #2 was resumed and completed. Similarly, I/O request #2 did not complete in frame #7; it was preempted and subsequently completed in frame #0.

Even though I/O requests #1 and #2 are not completed in their designated minor frame, the major frame requirements are met. That is, I/O requests #1, #2, #3, and #4 fulfill their respective 12.5 hz., 25 hz., 50 hz., and 100 hz. I/O requirements. This scheduling example is illustrated in Figure 5-62.


```

CREATE_I/O_REQUEST(  IOR #1,
                      RATE OF IOR => R1;
                      EXECUTION_FRAMES => -1;
                      CHAIN_ARRAY,
                      EXECUTION_TIME => 1.3 ms );

CREATE_I/O_REQUEST(  IOR #2,
                      RATE OF IOR => R2;
                      EXECUTION_FRAMES => -1;
                      CHAIN_ARRAY,
                      EXECUTION_TIME => 2.2 ms );

CREATE_I/O_REQUEST(  IOR #3,
                      RATE OF IOR => R3;
                      EXECUTION_FRAMES => -1;
                      CHAIN_ARRAY,
                      EXECUTION_TIME => 1.9 ms );

CREATE_I/O_REQUEST(  IOR #4,
                      RATE OF IOR => R4;
                      EXECUTION_FRAMES => -1;
                      CHAIN_ARRAY,
                      EXECUTION_TIME => 2.1 ms );

```

Figure 5-59. I/O Requests for Example #1

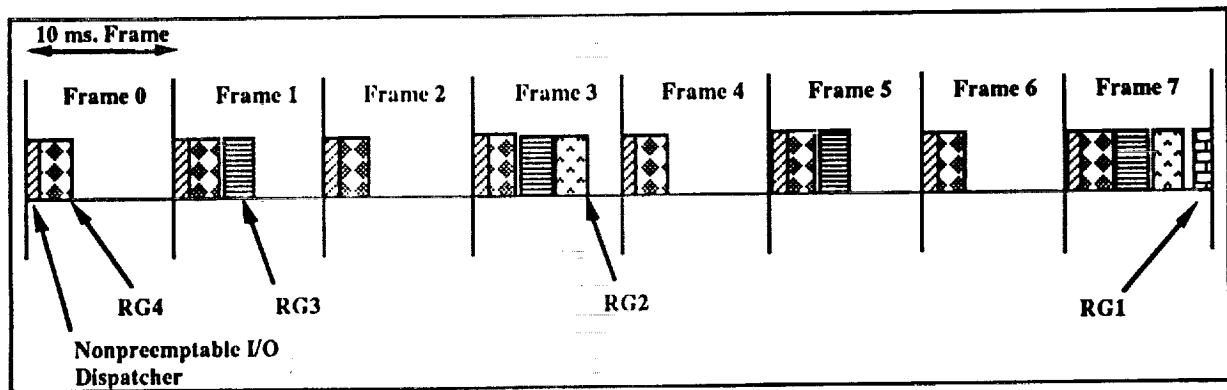


Figure 5-60. Example #1

```

CREATE_I/O_REQUEST(  IOR #1,
                     RATE OF IOR => R1;
                     EXECUTION FRAMES => -1;
                     CHAIN_ARRAY,
                     EXECUTION_TIME => 4.0 ms );

CREATE_I/O_REQUEST(  IOR #2,
                     RATE OF IOR => R2;
                     EXECUTION FRAMES => -1;
                     CHAIN_ARRAY,
                     EXECUTION_TIME => 3.2 ms );

CREATE_I/O_REQUEST(  IOR #3,
                     RATE OF IOR => R3;
                     EXECUTION FRAMES => -1;
                     CHAIN_ARRAY,
                     EXECUTION_TIME => 3.7 ms );

CREATE_I/O_REQUEST(  IOR #4,
                     RATE OF IOR => R4;
                     EXECUTION FRAMES => -1;
                     CHAIN_ARRAY,
                     EXECUTION_TIME => 3.6 ms );

```

Figure 5-61. I/O Requests for Example #2

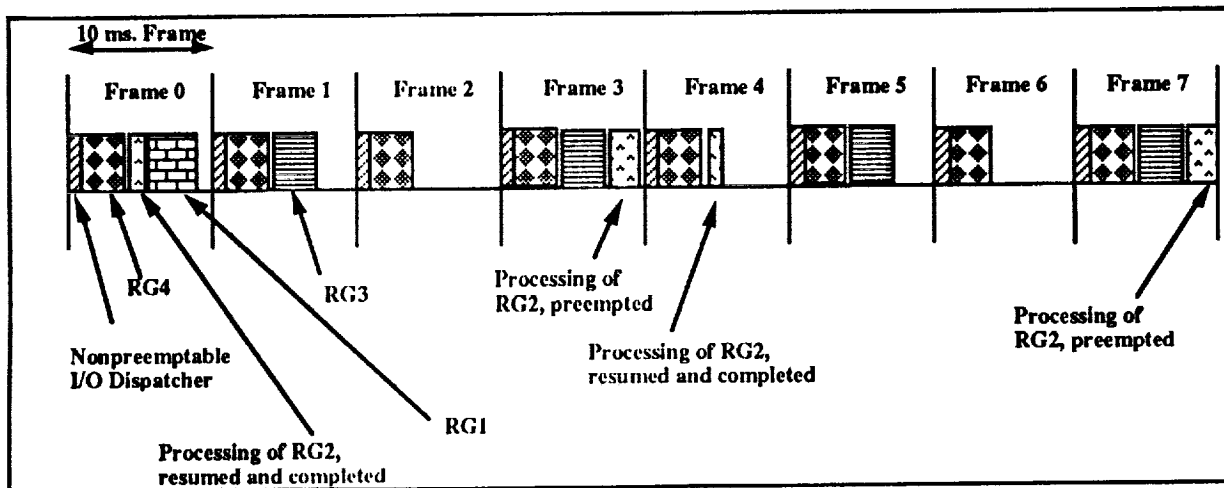


Figure 5-62. Example #2

6. Fault-Tolerant Data Bus

The fault-tolerant data bus (FTDB) is a local-area network designed around the same principles of Byzantine resilience as the AFTA. The FTDB is a highly reliable end-to-end communication system interconnecting the AFTA, other fault-tolerant computers, the Silicon Graphics display processor, the Merit Technologies MT-1 VME system, the real-time AI system, sensor and image processors, and flight and engine controls.

6.1. Objective and Approach

The objective of the fault tolerant data bus is to provide an optimal internetworking system between simplex and redundant processing sites. The approach taken in this report to develop such a system is to first identify requirements to which the FTDB should conform. Next, architectural options for the FTDB are described and evaluated with respect to figures of merit. Promising options are described in greater detail as a proposal for an FTDB architecture. Finally, a development plan for execution under the Detailed Design and Brassboard Fabrication phases of the AFTA program is described.

The following FTDB architectural options are investigated.

- broadcast buses
- token rings
- circuit switched network
- packet switched network
- fiber optic networks
- authentication protocols

The FTDB architectural options are evaluated according to the following figures of merit.

- bandwidth
- latency
- determinism
- compatibility with applicable standards
- fault tolerance
- topological flexibility
- complexity
- development cost/effort/risk

The following standards are investigated and considered for use in the FTDB.

- AIPS Inter-computer network
- JIAWG high-speed data bus
- SAVA high-speed data bus
- FDDI
- SAFENET II

A design for the FTDB architecture is presented based on the findings of the architecture survey. The conceptual design describes an end-to-end communication system for use between the AFTA, other fault-tolerant architectures, and simplex sites. A plan showing the development of a prototype FTDB for the AVRADA hotbench is described.

6.2. Fault-Tolerant Data Bus Requirements

The anticipated requirements for the fault-tolerant data bus are given below. Potential FTDB implementations are evaluated based on their adherence to these requirements. The requirements given below represent goals based on the anticipated needs of critical real-time systems. Some requirements may not be achievable due to time constraints, money constraints, or standards restrictions.

6.2.1. Packet Requirements

The requirements described in this section detail restrictions on packets in the FTDB.

6.2.1.1. Word Length

The FTDB shall incorporate a basic word length of 8 bits.

6.2.1.2. Packet Length

The FTDB shall allow packets to be any length between 1 basic data word and 2048 basic data words.

6.2.2. Network Control Requirements

This section details requirements for media access control, station addressing, and flow control.

6.2.2.1. Access Control Modes

Access to the FTDB shall be controlled using a distributed and symmetric access protocol. A station on the FTDB shall be able to obtain access to the network within an a priori determined latency.

6.2.2.2. Address Modes

The FTDB shall support the following addressing modes:

Physical addressing-The FTDB shall support physical addressing of up to 65535 stations.

Logical addressing-The FTDB shall support logical addressing of up to 65535 virtual station groups.

Broadcast addressing-The FTDB shall support broadcast addressing of all stations attached to the same physical FTDB.

6.2.2.3. Uncontrolled Transmit Inhibit

Each station on the FTDB shall provide a mechanism to prevent uncontrolled transmission (babbling). Transmission shall be inhibited if a continuous transmission exceeds 1.1 times the maximum message length. Transmission shall also be inhibited if a station exceeds its allotted transmission frame.

6.2.2.4. Flow Control

The FTDB shall implement a flow control mechanism so that a station with full receive buffers will prevent packets from being delivered to that station. Packets must not be lost due to a receive buffer full condition.

6.2.3. Network Function Requirement

This section describes the functions that the FTDB provides to subscribing stations.

6.2.3.1. Broadcast and Multicast Functions

The FTDB shall support broadcast and multicast functions. Only stations connected to the same physical medium as the source of the broadcast or multicast are required to receive the packet.

6.2.3.2. Periodic and Aperiodic Transfers

The FTDB shall provide both periodic and aperiodic packet transfers. Periodic transfers will request a fixed bandwidth allocation. The FTDB shall guarantee the bandwidth allocation to all periodic transmitting entities.

6.2.3.3. Packet Ordering

The FTDB shall deliver packets to station members, and to members of a multicast or broadcast group in the same relative order that the packets were transmitted.

6.2.3.4. Station Identification

It shall be possible for any station on the FTDB to determine if an expected station is active.

6.2.4. Topology and Architecture Requirements

This section describes requirements on the topology and architecture of the FTDB

6.2.4.1. Growth

The FTDB shall permit the addition or deletion of stations to an existing network. This addition or deletion shall not require modifications to either hardware or software of any station which does not communicate with the station in question.

6.2.4.2. Topology

The topology of the FTDB shall support from 2 to 100 stations on a single physical medium. The topology shall not restrict the physical or logical location of a station.

6.2.4.3. Station Insertion and Removal

The insertion or removal of a station shall not disrupt network traffic on an active network for longer than 1 second.

6.2.4.4. Bridges for Interconnected Buses

The architecture of the FTDB shall not preclude the use of bridges or gateways between FTDBs. Bridges and/or gateways do not need to support all traffic between FTDBs. In particular, the following are NOT required of bridges/gateways:

- Broadcasts and multicasts.
- Periodic data transfers.
- Deterministic latency.

6.2.5. Physical Requirements

This section describes requirements of the physical elements interconnecting stations on the FTDB.

6.2.5.1. Serial Transmission

The FTDB shall be implemented using a serial bus. All information, including data, clock, address, and control signals, must be capable of being placed on a single transmission medium (e.g., twisted pair, coaxial cable, fiber optic cable, etc.)

6.2.5.2. Media Support

The FTDB shall be compatible with any serial transmission media, including, but not limited to, fiber optic cables, twisted pair wire, or coaxial cable.

6.2.5.3. Electrical Isolation

There shall be no DC coupling between any two stations on the FTDB. The FTDB shall provide for at least 1000 volts of common-mode voltage rejection between stations.

6.2.5.4. Station Separation

The FTDB shall provide for separation of up to 1000 meters between stations on the same physical medium.

6.2.6. Fault Tolerance Requirements

This section describes requirements on the FTDB necessary to ensure the high reliability of the system.

6.2.6.1. Packet Delivery

Delivery of packets from a source to a destination in the FTDB shall be reliable. Packets shall not be lost due to single network faults, flow control, or collisions. The FTDB shall not require a retry mechanism to ensure packet delivery through an unreliable medium.

6.2.6.2. Synchronization

The FTDB shall provide a mechanism for stations to synchronize with other stations on the FTDB.

6.2.6.3. Source Congruency

The FTDB shall provide a mechanism for delivering packets transmitted by a simplex computing site to a redundant computing site such that the members of the recipient computing site receive bitwise identical copies of the packet. The FTDB shall correctly implement transmission between stations of simplex, triplex, and quadruplex redundancy levels to support Byzantine resilience.

6.2.6.4. Connectivity

An FTDB implementation shall provide sufficient connectivity so that multiple independent paths are provided between any two stations on the network. These independent paths shall have no common element. This connectivity may be provided with either multiple media layers or with a sufficiently interwoven network.

6.2.6.5. Station to Network Interface

The network interface unit (NIU) which connects a station to the FTDB network shall not permit a single station member to disrupt a path on the network.

6.2.6.6. Redundancy

The FTDB shall provide sufficient redundancy to allow reconfiguration around any fault, given that the fault is detected.

6.2.6.7. Station Redundancy

The FTDB shall support station redundancy levels of simplex, triplex, and quadruplex. The FTDB shall deliver data from a simplex to a fault-masking group (triplex or quadruplex) such that each member of the fault-masking group receives bitwise identical copies of the data.

6.2.6.8. Error Detection

A receiving station shall be capable of detecting errors in transmission. Upon error detection, it must be possible for the receiving station to unambiguously select a correct copy of the packet from a set of multiple copies without requesting retransmission of the packet. The error detection mechanism must be designed so that the likelihood of an undetected error is sufficiently small.

6.2.6.9. *Diagnosability*

The FTDB shall provide the capability to monitor the network and to detect any single fault.

6.2.6.10. *Self-Test*

The network interface unit (NIU) must provide sufficient independent self-test capability to ascertain the level of NIU functionality. The built-in self test capability shall have a coverage of at least 90%.

6.2.6.11. *Byzantine Resilience*

A single, arbitrarily behaved media, station member, or NIU fault shall not affect the overall operation of the FTDB. All packets must be delivered to their destinations in the presence of a single undetected, unreconfigured fault. After successful detection and reconfiguration of a fault, the reliability of the network must be returned to its original state, with the exception of the station(s) directly affected by the fault.

6.2.6.12. *Fault Isolation and Containment*

A fault in a NIU, station member, or interconnecting link shall not cause a fault in any other NIU, station member, or interconnecting link.

6.2.7. *Performance Requirements*

The following section describes the FTDB performance requirements.

6.2.7.1. *Message Priorities*

The FTDB shall provide for the following message priorities:

Priority S - Synchronous data exchange. The latency of a synchronous message must be guaranteed to be less than an a priori determined value. Synchronous data exchanges are the highest priority data. Any synchronous data messages enqueued in a transmitting station shall be transmitted before messages of any other priority level.

The FTDB shall also support four additional levels of message priority for normal data exchanges, named Priority 1, Priority 2, Priority 3, and Priority 4. The lowest numbered

priority level, Priority 1, is the highest priority of the normal data exchanges. Normal data exchanges are lower priority than Priority S.

A transmitting station shall transmit all messages of a given priority before messages of a lower priority are transmitted.

Only normal data exchanges must be supported across bridges or gateways.

6.2.7.2. Network Bandwidth

The FTDB shall provide a useable data transmission rate of at least 100 Mbits/second.

6.2.7.3. Initialization Time

The FTDB shall be available for packet transmission between the first two active stations on the network within 1 second of activation of the second station.

6.3. FTDB Architecture Study

This section discusses some of the architectural options for the FTDB. These options include topology, media technology, media access control protocols, and reliability enhancements. This section is not meant to be a comprehensive study of all possible options for the FTDB. Instead, only options which were deemed to have merit for applications in critical real-time environments were considered.

6.3.1. Broadcast Buses

A common type of physical network topology is the broadcast bus. An example of the broadcast bus topology is shown in Figure 6-1. Many examples of the broadcast bus exist, including IEEE 802.3, IEEE 802.4, and MIL-STD-1553. A common characteristic of these systems is that every node on the bus receives all data transmitted over the network. Each node selectively records data based on an address. If the address presented does not match the address the node is programmed to look for, the node ignores the data.

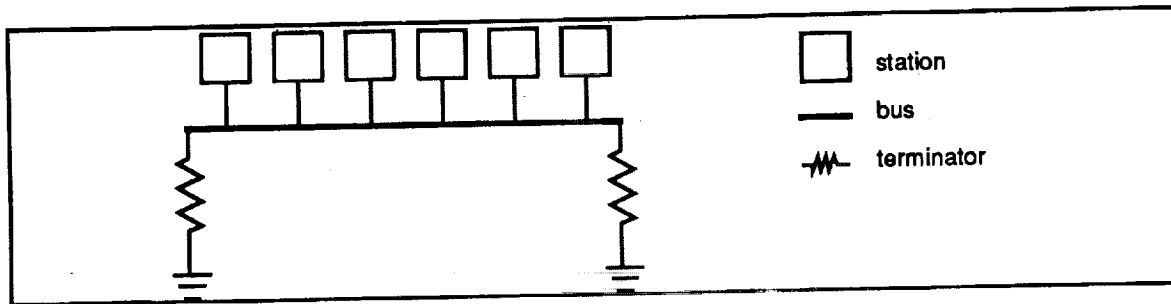


Figure 6-1. Broadcast Bus Topology

Only one node at a time is allowed to transmit data on the bus. Media access control is necessary to prevent the presence of two active transmitters. Many schemes exist for media access control on buses, including carrier sense, token passing, and centralized arbiters.

The IEEE 802.3 LAN [IEEE8023] is an example of a network that uses carrier sense for media access control. The exact protocol used by 802.3 (and Ethernet, on which 802.3 is based) is carrier-sense multiple access with collision detection (CSMA/CD). The process of transmitting data on the bus begins with the station checking the bus to make sure the media is clear. If so, the station begins transmitting. During transmission, the station monitors the bus, and if a collision with another transmitting station is detected, the station retries the broadcast. The time to retry is determined by a pseudo-random number to minimize the possibility of repeated collisions between stations. A variation of the CSMA/CD protocol is carrier-sense multiple access with collision avoidance (CSMA/CA).

While the carrier sense media access protocols are very common, they have some limitations. First is non-determinism. Slight differences between network implementations may favor one transmitter over another in a particular arbitration instance. Also, the bandwidth available to a transmitter may vary widely depending on other network traffic. In a system with varying bursts of data, it may be difficult for a transmitter to obtain the network regularly. Since regular, periodic data transfers are characteristic of real-time systems, carrier sense protocols may not be appropriate for a real-time system. Finally, a babbling station can monopolize the bus, preventing other stations from communicating.

A second common media access control is the token passing protocol. Buses which implement a token passing protocol are usually referred to as linear token passing buses (LTPB), an example of which is IEEE 802.4 [IEEE8024]. Each station on a LTPB is arranged in a virtual ring. A token is passed between stations on this virtual ring. A station is only allowed to transmit when it possesses the token.

The token passing protocol solves many of the deficiencies of carrier sense protocols related to real-time systems. Most token passing protocols use a token rotation timer to ensure that the token is delivered to each station within a fixed period of time. This token rotation time can be tailored to the periodic data transfer characteristics of the real-time system, thus ensuring that the deterministic data transfer needs of the system are met.

The MIL-STD-1553 [MIL-STD-1553] data bus uses a centralized bus arbiter to control access to the bus. A 1553 network has a single controller device. Initially, only the controller is allowed to transmit on the bus. The transmitter selects other nodes on the network to allow them to drive the bus for fixed periods of time. When the remote node is finished transmitting, bus ownership is returned to the bus controller, which then selects another node to drive the bus.

Centralized media access control has the advantage of simplicity over most other media access protocols. However, it also has many limitations. Since the media access control is centralized in the bus controller, the bus controller becomes a single point of failure. Also, interrupt delivery from a remote node to the bus controller is difficult. Thus, each node on the network must be polled by the controller to determine if it has data to transmit. This polling could be significant for high iteration rate control systems, reducing the available bandwidth for actual data transfers.

Broadcast buses are typically built using electrical components, such as twisted pair wire or coaxial cable, with transformer coupling for isolation. Optical devices do not lend themselves to the construction of broadcast buses. Fiber optics are inherently unidirectional devices, whereas electrical wires are easily made bidirectional. Typically, multiple fiber optic splitters/mixers are required to build a broadcast bus with fiber optics. The optical losses associated with the splitters/mixers become significant for a very small number of network connections. One of the advantages of the broadcast bus topology is the simplicity of connecting a station to the network. This advantage is lost when fiber optics are used in place of electrical components.

The fault tolerance of a simplex broadcast bus is not good. A single station or link fault can disrupt the entire network. There are few remedies for these situations except to physically remove the faulty station or fix the broken link.

Some broadcast buses have been designed for fault-tolerance. These designs incorporate redundancy in the form of spare links, redundant media layers, or both. A bus

with spare links built into the network is reconfigured by switching in a spare link to replace a faulty link, regrowing the original bus topology. Redundant media layers are usually used in an active/standby mode. One layer is used until a fault is detected, at which point all stations switch over to the secondary layer. The MIL-STD-1553 bus is an example of a bus with active/standby layers. Alternatively, three or more redundant layers can be used to transmit redundant copies of data, which are voted upon reception. The AIPS intercomputer network [CSDL9214] uses both redundant, voted media layers and spare links to achieve high reliability.

6.3.2. Token Rings

Token ring networks are another common network topology. Token rings are constructed using electrical components or fiber optics. The inter-station links on a token ring network are unidirectional, making fiber optics more viable for a token ring system than for a broadcast bus. A diagram of a simplex token ring network is shown in Figure 6-2.

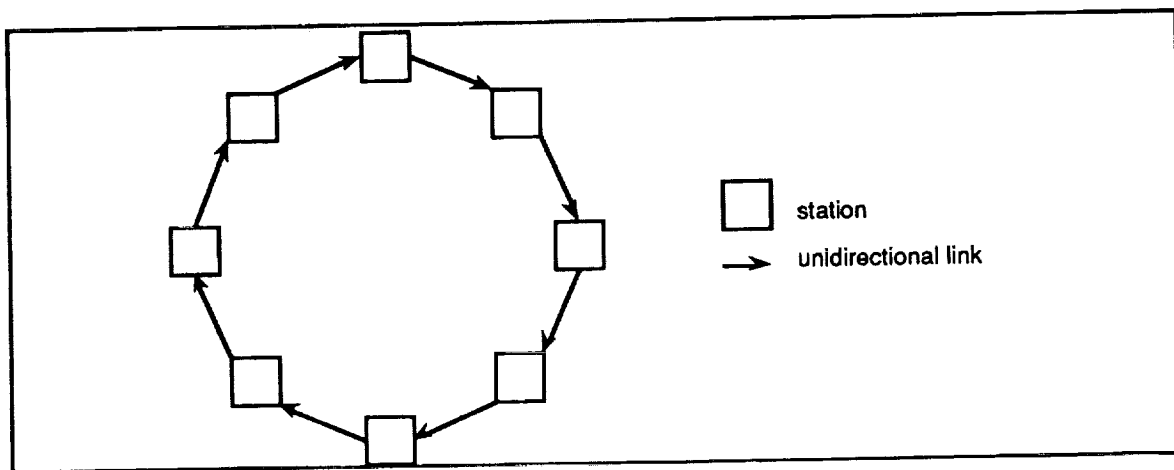


Figure 6-2. Token Ring Topology

Token ring networks use a token passing protocol for media access control. The deterministic nature of the token passing protocol makes it suitable for use in real-time systems.

A single ring network is highly susceptible to interruption from faults. Any single station or link failure will disrupt the network. Even passive station faults, such as loss of power, are not tolerated since such failures disrupt the passing of the token. Most token rings have mechanisms for regenerating a lost token. However, full network functionality can be regained only by fixing the broken station or link.

Token passing protocols assume the ring is contiguous, so ring topologies must be reconfigured around faults to enable the tokens and data to make a complete circumnavigation of the ring. Ring regeneration options include chordal rings, dual counter-rotating rings, and station bypass.

The chordal ring approach requires redundant links which bypass nodes and links on the primary ring. A fully braided ring, such as that shown in Figure 6-3, can be implemented for arbitrary reconfigurability, or bypasses can be placed only across nodes which are expected to fail most often or which are considered non-essential. When a failure is detected, a redundant link is switched in to replace the broken node or link. The redundant link re-forms the ring, and network traffic can proceed around the ring uninhibited.

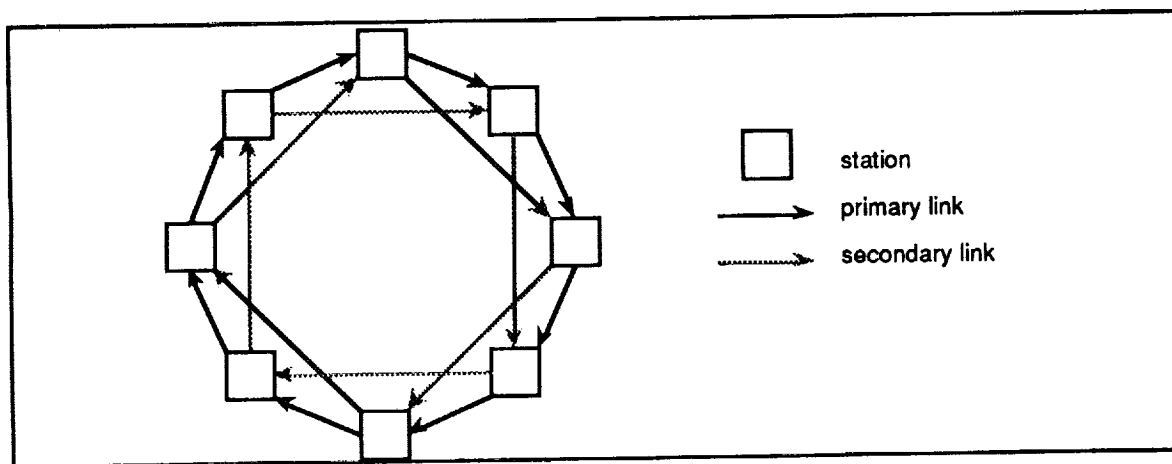


Figure 6-3. Fully Braided Chordal Ring

A redundant counter-rotating ring, shown in Figure 6-4, is built with bidirectional links between each station and the two adjacent stations. The links traversing the ring in one direction are used as the primary ring, and the links in the other direction make up the secondary ring. A fault on the primary ring causes all stations to switch over to the secondary ring. The network can reconfigure around a fault on the secondary ring by building a loopback ring with pieces of both the primary and the secondary ring. However, depending on the location of the fault, certain stations may be isolated from other stations on a loopback ring.

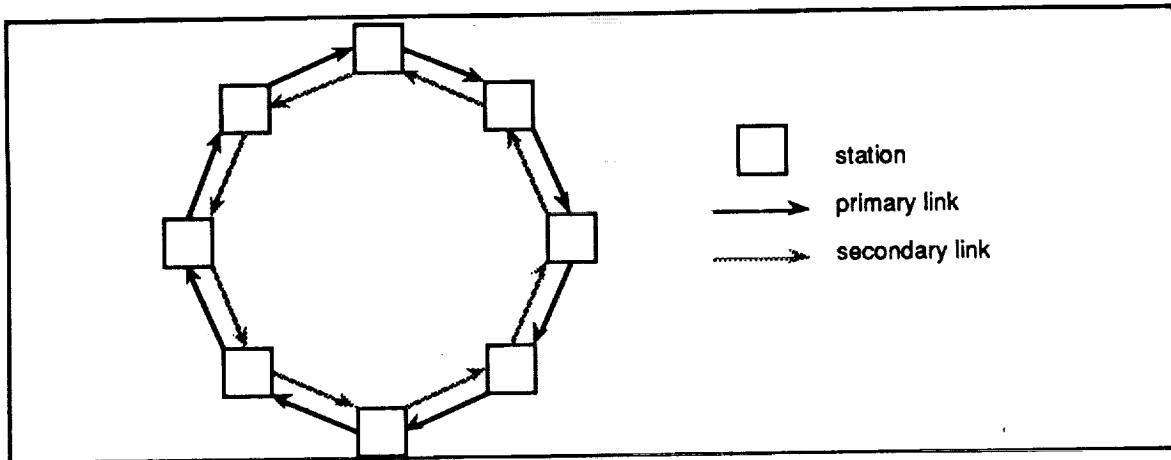


Figure 6-4. Dual Counter-rotating Ring

Station bypass is a mechanism for reconfiguring around passive station faults that prevent forwarding of tokens or data. Station bypass allows a station to voluntarily bypass its connection on the ring. The station bypass switch is usually designed so that if a station is powered down, the network inputs to the station are shunted to the outputs. Station bypass only works for passive faults since a maliciously failed station will refuse to shunt the station bypass switch.

6.3.3. Circuit Switched Network

A circuit switched network provides guaranteed bandwidth between any two stations. Bandwidth is allocated by establishing a connection between two interacting stations. Part of the connection establishment primitive is a bandwidth request. A fixed percentage of the bandwidth of an internode network link is allocated, either by time or frequency domain multiplexing, when a connection is established. The network assumes that the transmitting station will use all of its available bandwidth. If the connection is under-utilized, the unused bandwidth is wasted.

A possible topology for a circuit switched network is shown in Figure 6-5. Virtually any interconnection topology is possible. The number and connectivity of links in the network varies depending on the expected traffic between each possible pair of nodes.

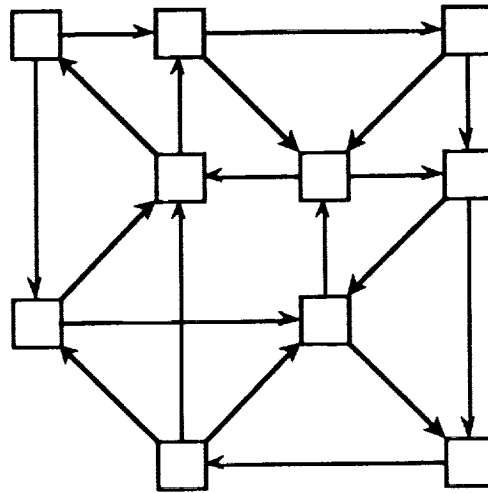


Figure 6-5. Example Circuit Switched Topology

The latency of circuit switched networks is very good. Typically, once a connection is established, the latency imposed by intermediate nodes in the path is minimal. As an example, a cross-country telephone connection typically has a latency of less than 30ms [Tan88].

All communications on a circuit switched network must occur within the allotted bandwidth slot. Thus, a connection must be established before a station can communicate with any other station. Circuit-switched networks work best with connection-oriented communication protocols; they do not work well with datagram protocols. A station could establish connections with all other stations with which it might want to communicate. However, this action results in a large amount of wasted bandwidth unless communications are very regular. Alternatively, a station could establish a connection for each desired communication, transmit the message, and destroy the connection. This procedure would minimize wasted bandwidth; however, message communication would be delayed by the overhead of repeated connection/disconnection operations. Also, broadcast and multicast communications are not possible on a circuit switched network.

Circuit switched networks are typically used in situations where a steady stream of data must be delivered between distinct points on the network. An example is a signal distribution network. Digital or analog signals typically have constant bandwidth requirements, therefore a connection can be established for transmission of a signal without severe bandwidth wasting. A system with highly irregular bandwidth requirements, such as a typical data processing system, are unsuitable for circuit switched networks. While a

real-time system has constant components to the network utilization model, these components are a small part of the total data communication.

The circuit switched network is easily built with redundant links for enhanced reliability. Also, the network can be designed so that a node will only affect nodes and links to which it is directly connected. A node cannot affect another node which is more than one network hop away. Thus, mutually exclusive paths between nodes can be constructed with a circuit switched network without the necessity of redundant media layers.

6.3.4. Packet Switched Network

An alternative to static bandwidth allocation in a circuit switched network is to dynamically allocate bandwidth based on instantaneous network usage. A network using this method of bandwidth allocation is called a packet switched network. Each packet arriving at a node is either delivered to a station within the node or forwarded to an output port on the node. The choice of output port is made based on the eventual destination of the message and the current utilization of the output ports. The network utilization may vary widely, so each node must be able to buffer messages until the network utilization declines enough to allow the message to be sent. Because of this property, packet switched networks are sometimes called store-and-forward networks.

Packet switched networks are typically used for wide-area networks (WAN) across countries or even the entire globe. Wide-area packet switched networks usually provide interconnections of local-area networks of other topologies such as buses or rings. The Internet is one well-known example of this type of packet switched network.

The packet switched network has some limitations in local area network applications. Packet switched networks usually require complex distributed routing algorithms, and each node must apply the algorithm to each incoming packet, increasing packet latency. Broadcasts and multicasts over packet switched networks are difficult, since there is no convenient way to determine whether or not a node has received a broadcast or multicast message. Packet ordering is also a problem if the network uses datagram routing. Because two packets may follow different routes, the packets may arrive at the destination in the reverse order that they were transmitted in. Virtual circuit routing solves the packet ordering problem, since all packets are forced to follow the same route. However, virtual circuit routing requires the establishment of a connection before communication can take place.

Finally, because of the complexity of a packet switched network, the validation of the network to guarantee deadlock-free operation is difficult.

Packet switched networks can be built with the same topological freedom afforded circuit switched networks. Thus, highly reliable packet switched networks can be built with only a single media layer, provided that there are multiple mutually exclusive paths between any two nodes on the network.

6.3.5. Fiber Optic Networks

The use of fiber optics for computer networks is becoming increasingly widespread. Fiber optics provide many advantages over traditional electrical interconnect such as coaxial cable or twisted pair wire.

The advantages of fiber optics over copper media are numerous. Fiber is not susceptible to electromagnetic interference (EMI). Also, because the fiber does not radiate any EMI of its own, it is more resistant to eavesdropping. Fiber provides excellent electrical isolation between systems, an important property for fault-tolerant systems. The bandwidth capacity of fiber is considerably higher than coaxial cable, particularly over long distances.

Most of the disadvantages of fiber optics are related to cost. A point to point communication link using fiber optics can cost 10 to 50 times as much as a comparable system using coaxial cable. Building a broadcast bus using fiber optics is difficult because many fiber optic splitters are required. A simple 4 x 4 fiber optic splitter costs about \$500, whereas a splitter using coaxial cable, a few T-splitters, and BNC connectors costs about \$10. An electrical connection can be shunted using a simple relay from Radio Shack, but an optical bypass switch is very expensive.

6.3.6. Authentication Protocols

The solution to the Byzantine Generals Problem [LSP82] prescribes a method, called source congruency, for guaranteeing that data delivered to each member of a redundant processing site, or fault-masking group (FMG), is always in agreement, even in the presence of a single random fault of arbitrary behavior. Additionally, the source congruency guarantees that, if the original source of the data is non-faulty, the data delivered to each member of the FMG is valid. Traditional implementations of the source congruency algorithm use at least three unsigned copies of a message exchanged in a two

round exchange pattern to provide sufficient redundancy in the event of a single random fault.

An alternative to the source congruency algorithm using triplicated unsigned messages is to attach a signature to each message. For a source congruency algorithm using signed messages, the following conditions must be satisfied:

- A faulty node can not interfere with the communication between two other nodes, unless the communication path involves the faulty node.
- The receiver of a message knows who sent it.
- The absence of a message can be detected.
- Any alteration of the message can be detected.
- The signature of a functioning station cannot be forged.
- Any node can verify the authenticity of a signature.

When a node receives a message, it checks the authenticity of the signature. If the signature is valid, the node assumes that the contents of the message are the contents the sender intended. Thus, interference by intermediate nodes is ruled out. If an intermediate node corrupts the content of the message, the receiving node would detect the corruption and declare the signature invalid.

The message signature can either be message specific or non-message specific. Traditional signed paper documents, such as checks, wills, or contracts, use a non-message specific signature. Non-message specific signatures assume that the exact pattern of the signature is difficult to duplicate. Any attempt to duplicate the signature by tracing, photocopying, etc. is easily detected. However, in a computer system, any bit stream is easily copied. A system need only contain enough memory to store the signature stream to be able to reproduce the signature pattern. The nature of computer systems prevents this signature copy from being distinguished from the original.

The solution to the signature copying problem is to use message specific signatures. A signature is calculated as a function of the message. Thus, each message has a different signature attached to it. To verify the signature, the receiving node applies another function to the message to determine if the signature is valid.

The functions used to generate and authenticate the signature must be chosen to minimize the likelihood of a successful forgery attempt. Ideally, no node except the sending node should know how the signature was generated. The receiving node should be able to

test the authenticity of the signature without requiring knowledge of the signature generation.

Functions with the properties described above can be found in the field of public key cryptography. Public key cryptography uses two functions, $E()$ and $D()$. $E()$ is a public encryption function, of which everyone has knowledge. The decryption function, $D()$, is private and known only to the receiver. The function $D()$ cannot be easily deduced from knowledge of $E()$. Anyone can encrypt a message using $E()$, but only the intended receiver can decrypt an encrypted message. Thus, an intercepted message cannot be decoded by an unauthorized station. The encryption/decryption function pairs have the property:

$$D(E(M)) = M$$

For authentication protocols, a slightly different operation is needed. The source station keeps the private key to generate signatures, and the destination stations use the public key to test the authenticity of the signature. The source must keep the private key so that no other station can forge a signature, and the destinations must have the public key so that any one of them can verify the authenticity of a signature. Note that the message itself is not necessarily encrypted for authentication protocols; if such encryption is desired, another private/public key pair can be used as described above. However, the keys for encryption and the keys for authentication are distinct.

Since the private key is applied before the public key, only isomorphic function pairs, with the following property, are suitable for use in authentication protocols:

$$E(D(M)) = M$$

The signature for a message is generated by applying $D()$, the private key function, to the original message. Then the message and the signature are transmitted to the receiving node. The receiver applies $E()$, the public key function, to the signature, and if the result matches the original message, the signature is valid.

In practice, a signature calculated from the message using the procedure outlined above is extremely unwieldy. The signature, $D(M)$, is at least as long as the original message, M , since M can be regenerated from it. Thus, half of the communication bandwidth is consumed by the signature transmission. However, the signature does not have to be the same length as the message. For the AFTA project, a fixed length signature of 64 bits is sufficient to meet the reliability requirements [Gal90]. A system using fixed length

signatures requires a common function, $C()$, to be applied to the message. The sender generates the signature by applying the following:

$$S = D(C(M))$$

The receiver validates the signature by applying the following test:

$$C(M) == E(S)$$

A possible implementation of the common function is the cyclic redundancy check (CRC) function. The CRC is widely used as an error detecting code for data communication. The detection of random, arbitrarily behaved errors with CRCs is very good, thus an authentication protocol that uses CRCs for error detection is reasonably secure against forgery. In addition, the CRC function can be tailored to detect some expected error patterns, such as burst errors or double-bit errors.

One possibility for the public/private key functions is a system using modular inverses. Modular inverses have the following property:

$$M \cdot_2 N = 1$$

where M and N are both 64 bit integers. Modular inverses are similar to multiplicative inverses in the set of real numbers, except that modulo multiplication is used instead of arithmetic multiplication. The modular inverse scheme is not cryptographically secure, since the calculation of modular inverses is relatively straightforward for a human cryptographic expert. However, the reasonable assumption is made that a station will not fail in a manner that turns it into a cryptographic expert with the ability to calculate modular inverses.

The signature generation and authentication procedures would be as follows: the sender multiplies the 64 bit CRC by the 64 bit private key integer using modulo-64 multiplication, yielding another 64 bit number which becomes the signature. The receiver regenerates the CRC by multiplying the 64 bit signature by the public key, the 64 bit modular inverse of the private key. This CRC is compared by the receiver with a locally generated CRC on the received message to test the authenticity of the signature.

Message specific signatures, such as those described above, prevent a node from saving another node's signature for use in forging a new message. However, an intermediate node can save and retransmit the message itself, complete with valid signature. The receiving node must be able to distinguish between bogus copies of a message from a

broken intermediate node and legitimate repetitions of the message from the original source. A solution to this problem is to force the message to contain a known varying component. Even a repeated message from the same node will never be exactly identical to previous copies. An example of a varying component is a sequence number. A receiving node knows what sequence number should be contained in the next message from a source node. If a message arrives from a node with an incorrect sequence number, the receiving node rejects the message as erroneous.

6.4. Existing and Proposed Standards

This section describes several networking standards, both existing and proposed, that are of interest. Many of these standards are designed for application in real-time military systems. Since the FTDB is targeted for military systems, these standards may have a great influence on the acceptance of the FTDB.

Each standard defines certain aspects of a local-area network for real-time applications. The scope of each standard varies. Some standards only define the physical and data link layers. Other standards define a complete end-to-end communication system with a fully specified protocol stack.

None of the standards presented below, with the exception of the AIPS IC network, will survive Byzantine faults without modification. However, most of these standards were developed with real-time system applications in mind. Thus, some of the techniques specified in these standards can be applied to the fault-tolerant data bus design.

6.4.1. AIPS Intercomputer Network

The Advanced Information Processing System (AIPS) developed at the Charles Stark Draper Laboratory [CSDL9214] includes the definition of an intercomputer, or IC, network. The IC network is designed to interconnect simplex, duplex, triplex, and quadruplex AIPS processing sites. The fault-masking processing sites and the IC network are designed to be Byzantine resilient.

A diagram of the AIPS IC network is shown in Figure 6-6. The network consists of three identical layers. Each layer is totally isolated from the other layers. Each member of a processing site only transmits on one layer, although it receives from all three layers. There is no cross-strapping between the layers except within a redundant processing site and on delivery of data from the network to a processing site (the latter is not shown in Figure 6-6

for clarity). Each layer forms a broadcast bus with media access controlled by a deterministic bus arbitration mechanism.

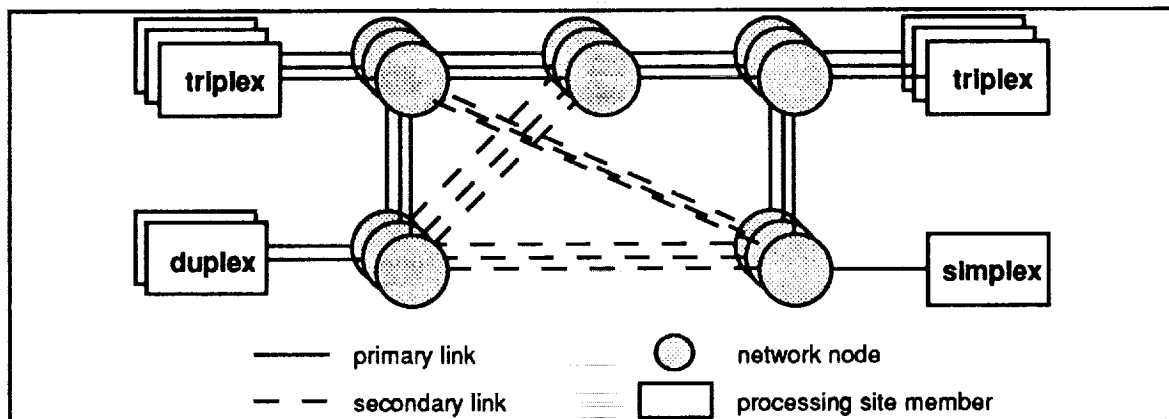


Figure 6-6. AIPS IC Network

Although the IC network acts as a broadcast bus, the network is actually implemented as a set of point to point links. Each layer contains more than enough links to form a bus to interconnect all nodes. The network manager selects a subset of the interconnecting links to form a bus. The remaining links in the layer are unused until a fault is detected, at which time the network manager switches in a redundant link to replace the failed link. The network configuration established by the network manager is not changed unless a fault is detected.

The AIPS IC network uses a media access scheme called the Laning poll [CSDL9214] to control access to the bus. The Laning poll guarantees that all members of a station contending for the network deterministically obtain or relinquish the bus. Each station is assigned a unique priority number which is used during bus arbitration to request access to the bus. The requesting station with the highest priority number obtains the bus.

6.4.2. SAVA High-Speed Data Bus

The Standard Army Vetrionics Architecture (SAVA) defines a high-speed data bus (HSDB, not to be confused with JIAWG HSDB) for interconnection of computing sites within the SAVA architecture. The SAVA architecture is designed for implementation inside ground vehicles, particularly battlefield vehicles such as tanks.

The current draft standard for the SAVA HSDB is [MIL-STD-344]. At this time, [MIL-STD-344] is still under development. Some of the characteristics discussed in this section were inferred from data in the draft standard and may be incorrect. The SAVA HSDB draft

specification defines only the physical and data link layers of the ISO/OSI protocol model. No mention is made in [MIL-STD-344] of any layers above the data link layer.

The SAVA HSDB allows interconnection of up to 32 nodes on a single physical LAN. The network uses transformer coupling to a twinaxial cable bused throughout the vehicle. A 12 MHz signaling rate using Manchester encoding is used to transmit data on the bus, thus an effective data exchange rate of 6 Mbits/sec is realized.

Access to the media is controlled by a token passing protocol. Tokens are not really captured for any length of time. Instead, a station can transmit on the bus only after a token is received, and the token is transmitted immediately following the message. Thus, each station is only allowed to transmit one packet per token reception.

Network fault-tolerance is specified by a dual-layer network. Each station is connected to both layers, thus the SAVA HSDB will not tolerate Byzantine faults without additional isolation between media layers. Stations are equipped with media bypass so that passive station faults can be tolerated.

6.4.3. JIAWG High-Speed Data Bus

The Joint Integrated Avionics Working Group (JIAWG) defines a high-speed data bus (HSDB) for interconnecting modules within the advanced avionics architecture (A3). The A3 architecture is targeted for application in the advanced tactical fighter (ATF), the advanced tactical aircraft (ATA) (cancelled), and the light helicopter (LHX) [J8701].

The current draft standard for the JIAWG HSDB is [J88N2]. At this time, [J88N2] is still under development. Some of the characteristics discussed in this section were inferred from data in the draft standard and may be incorrect. The JIAWG HSDB draft specification defines only the physical and data link layers of the ISO/OSI protocol model. No mention is made in [J88N2] of any layers above the data link layer.

The JIAWG HSDB uses an optical bus topology with token passing media access control. A virtual ring network is superimposed on the physical bus topology. The network uses Manchester encoding, so a maximum data transfer rate of 50Mbits/sec is obtained using a 100MHz signaling rate.

The JIAWG HSDB provides for network fault-tolerance by specifying dual redundant buses. While the HSDB as defined in [J88N2] does not describe the network topology, it

is highly likely that the dual buses are not sufficiently isolated for Byzantine resilience. Past experience indicates that unless a design specifically addresses the issue of Byzantine resilience (which JIAWG does not), the design cannot survive Byzantine faults without additional redundancy.

The JIAWG specification [J88N2] does not describe how to construct a broadcast bus using fiber optics.

6.4.4. Fiber Distributed Data Interface (FDDI)

The fiber distributed data interface (FDDI) was developed by the American National Standards Institute (ANSI) to satisfy increasing demands for high bandwidth local-area networks. The FDDI standard is gaining momentum as the next generation local-area network topology, complementing the current popular choice, Ethernet. FDDI is also recognized by the International Standards Organization (ISO) as a protocol for open system interconnect (OSI). FDDI is currently defined by three American National Standards, [ANSI139], [ANSI148], [ANSI166], and one draft standard, [X3T95]. Most FDDI implementations also use the logical link control specified by [IEEE8022].

FDDI defines two counter-rotating rings, a primary ring and a secondary ring. The primary ring is used unless a fault is detected, in which case the secondary ring may carry all or part of the network traffic. The two rings both connect to a single network interface. The dual ring design of FDDI allows reconfiguration around detected faults. However, because the two rings share a network interface, additional redundancy is required for Byzantine resilience.

The media access control in FDDI is managed by a token passing protocol. The token passing system guarantees a bounded latency for interstation communications. The FDDI token passing system defines synchronous and asynchronous bandwidth allocation. Synchronous bandwidth allocation is guaranteed to each station. Asynchronous bandwidth is taken from whatever is left over after all synchronous messages have been transmitted. The inclusion of synchronous bandwidth makes FDDI ideal for real-time systems where bounded, deterministic network response is necessary.

The specification of FDDI is divided into four major sections, corresponding to each of the four accepted and draft standards. These sections are the physical layer protocol, the physical medium-dependent layer, media access control, and station management.

The physical layer protocol (PHY) and physical medium-dependent layer (PMD) make up the ISO/OSI physical layer. The physical medium-dependent layer defines the hardware of the FDDI network, such as light wavelength, fiber diameter, cable plant dimensions, and optical transceiver characteristics. Physical characteristics not directly affecting interoperability, such as fiber sheathing, are not defined by PMD. The physical layer protocol defines the medium independent characteristics of the FDDI network. The scope of the PHY standard includes coding, symbol set, signaling protocol, and clock synchronization.

The media access control (MAC) and station management (SMT) reside in the data link layer of the ISO/OSI model. The media access control specification defines the token passing protocol with synchronous and asynchronous bandwidth, station physical, logical, and broadcast addressing, packet formats, and network initialization. The station management includes provisions for network configuration management, fault isolation and recovery, ring scheduling procedures, and station initialization.

An FDDI implementation also requires a logical link control (LLC) protocol. The LLC is responsible for delivering packets, or Protocol Data Units (PDU), to the appropriate higher level protocol stack. The current definition of FDDI in the four ANSI standards does not specify a logical link control. However, by convention, most FDDI implementations use the IEEE LLC [IEEE8022]. This LLC provides for delivery of packets between service access points (SAP). A source LLC specifies a SAP to which the LLC on the destination is to deliver the packet. The destination SAP usually specifies a protocol stack. Administration of SAPs is global, so that all systems using the IEEE LLC will properly recognize or ignore packets as they arrive at the station.

FDDI has been studied for use in other real-time systems, with the conclusion that real-time systems requirements can be met [Coh87].

6.4.5. SAFENET II

The Survivable Adaptable Fiber Optic Embedded Network, or SAFENET, is being developed by the Navy to satisfy intercomputer communications requirements for shipboard, aircraft, and ground-based systems. The current draft standard for SAFENET II is [MIL-HDBK-0036]. SAFENET II is intended to meet the needs of both interactive and real-time systems. The SAFENET II defines a complete end-to-end local-area network. The

entire ISO/OSI protocol stack, from the physical layer to the presentation layer, is defined by SAFENET II.

The design of SAFENET II is based on the FDDI specification. SAFENET II defines a new physical medium dependent (PMD) layer using militarized components. Each station is connected to the medium by a trunk coupling unit (TCU). The TCU contains an optical bypass switch with which a station can voluntarily bypass itself on the network. The TCUs are interconnected by fiber optic cable. All fiber optic cable junctions are constructed using fiber optic splices, except for station connections. A network station is allowed to connect to its TCU using a militarized fiber optic connector.

A station may connect to either one (single attach) or two (dual attach) of the SAFENET II rings. Since a station, consisting of one fault-containment region, may connect to both rings, the dual ring design is not sufficiently isolated to tolerate Byzantine faults. A dual attach station may listen on only one of the rings at a time. One ring is designated the primary ring, and is used until a fault is detected. Upon fault detection, all dual attach stations switch over to the secondary ring. If a fault is detected in the secondary ring, a new ring may be constructed using segments of the two original rings. All station bypass and ring reconfiguration operations assume that no Byzantine faults have occurred.

The SAFENET II specification also defines the higher layers of the protocol stack. Two protocol suites are specified: an OSI compliant protocol suite based on the Manufacturing Automation Protocol (MAP) and a lightweight protocol suite based on the Xpress Transfer Protocol® (XTP®) [PEI90120]. A SAFENET II station may implement either or both of these protocol suites.

6.4.6. Summary

Table 6-1 below summarizes the differences between the various physical and data link protocols discussed above. Several interesting conclusions arise from this table. First, most of the systems outlined use token passing media access control, suggesting that such a media access scheme is optimal for real-time systems. Also, all systems consider redundancy to enhance system reliability. However, only the AIPS IC network defines sufficiently isolated redundant network layers to tolerate Byzantine faults. Finally, the FDDI-based systems provide the highest bandwidth of any of the systems investigated. The nearest competitor is the JIAWG HSDB, which has only 50% of the FDDI bandwidth.

Standard Characteristic \	FDDI	SAFENET II	JIAWG HSDB	SAVA HSDB	AIPS IC
topology	dual ring	dual ring	dual bus	dual bus	triplex bus
media access control	token passing	token passing	token passing	token passing	Laning poll
signaling rate	125MHz	125MHz	100MHz	12MHz	2MHz
data rate	100Mbits/sec	100Mbits/sec	50Mbits/sec	6Mbits/sec	2Mbits/sec
signaling method	4B/5B/NRZI	4B/5B/NRZI	Manchester	Manchester	HDLC
physical medium	fiber optics	fiber optics	fiber optics	twinax cable	coaxial cable
reconfiguration mechanism	dual rings, bypass	dual rings, bypass	dual buses	dual buses, bypass	redundant links

Table 6-1. Comparison of Standards

These conclusions direct attention toward FDDI as an excellent candidate for use in the FTDB. The throughput of FDDI is higher than any other system investigated. Since FDDI uses fiber optics, inter-node isolation is excellent. The token passing media access for FDDI ensures deterministic bus access for real-time tasks. While FDDI does not provide sufficient isolation between its dual rings, additional redundancy in the form of multiple FDDI networks can be used to provide the necessary isolation. Finally, the FDDI standard itself is very stable. Several of the other standards, particularly JIAWG HSDB and SAVA HSDB, are very preliminary and implementation details are very sketchy. Neither of these systems is adequately specified in the current version of the standard to construct a working system. The FDDI standards, on the other hand, have been accepted by both ANSI and ISO (with the exception of the station management standard, which is in the final stages of acceptance), and working systems employing FDDI are available as off-the-shelf items.

6.5. FTDB Brassboard Design Proposal

This section describes the conceptual design for the AFTA fault-tolerant data bus. The brassboard FTDB design will be constructed as described by the FTDB brassboard development plan.

The FTDB design is based on the ISO/OSI model for data communications [Bla91]. The ISO/OSI model of the FTDB is shown in Figure 6-7. Only the lower four layers are described by the conceptual design; specifications for the upper layers are currently beyond the scope of the FTDB.

The physical and data link layers are based on the FDDI physical and data link layers. The network layer protocols are designed to handle the redundancy management and message authentication necessary to support Byzantine resilience. The transport layer

protocols provide several different data models for inter-processor communication. The transport protocols are integrated with the AFTA Ada run-time system. All redundancy management issues are hidden from the user by the transport and network protocols.

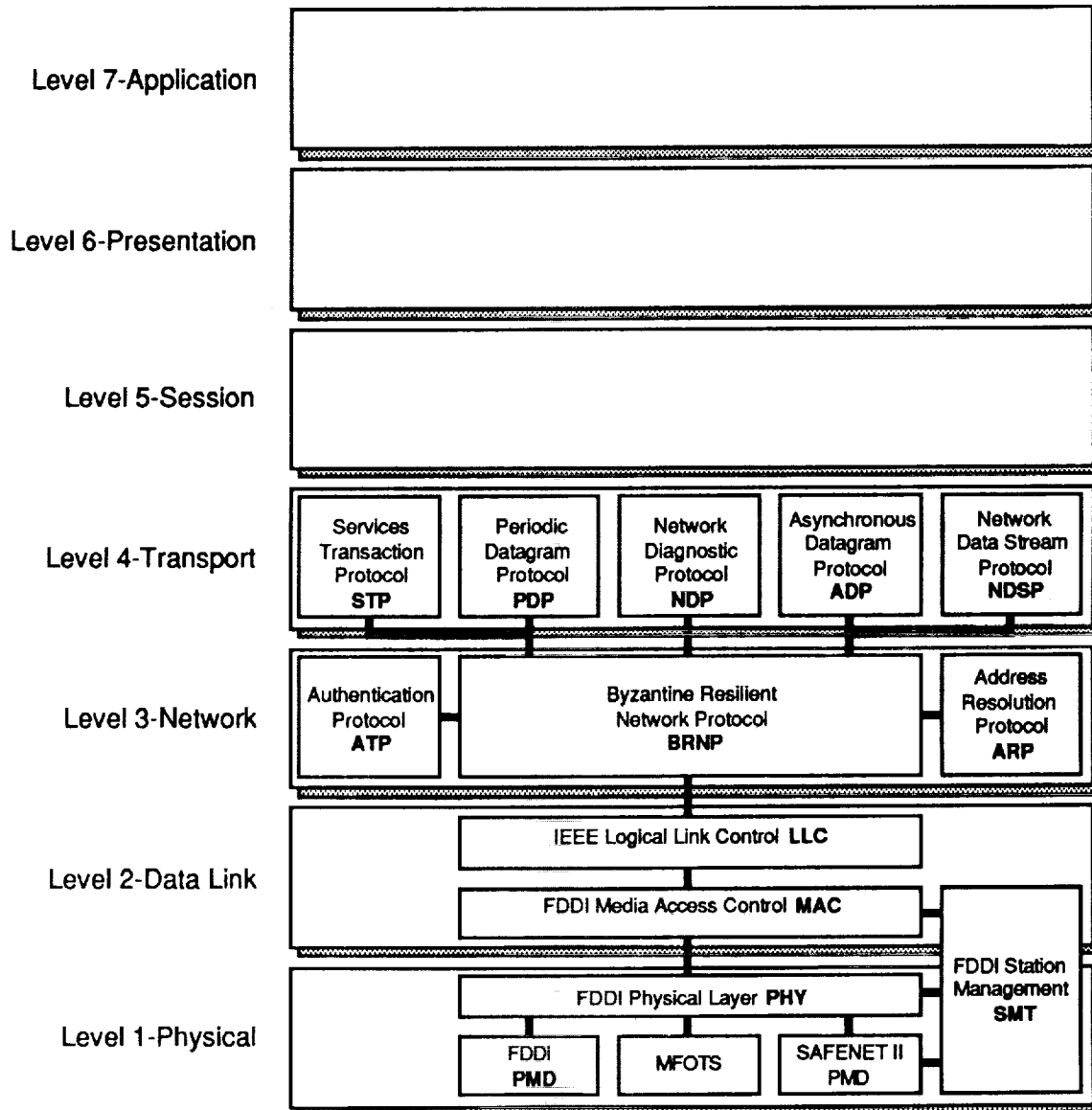


Figure 6-7. ISO/OSI Model of FTDB

The physical and data link layers are designed around the ANSI specifications for FDDI and the IEEE logical link control standard. The FDDI standard is rapidly gaining momentum as the next generation local-area network. Many existing network standards, such as MIL-STD-1553 and Ethernet, are widely used and well established, but the technological state-of-the-art is making these systems obsolete. Exotic technologies like gallium-arsenide (GaAs) make very high throughput (above 1 Gbit/sec) communication

possible; however, such technology is very new and standards based on this technology have not yet been established. The FDDI standards, at 100 Mbits/sec, represent an appropriate balance between high technology and standardization.

FDDI contains many features that are useful for real-time fault-tolerant systems, including high raw bandwidth, low latency, deterministic token passing media access, synchronous and asynchronous bandwidth, and network fault detection and recovery.

The 125 MHz signaling rate and the 4B/5B code with NRZI signaling of FDDI provides a raw bandwidth of 100 Mbits/sec, higher than other established standards in either the commercial or the military sector. A maximally configured FDDI network, with 1000 stations and 200 km of cabling, has a ring latency of 1.617 ms. A network of reasonable size for embedded applications, with 100 stations and 10 km of cabling, has a ring latency of 0.111 ms, slightly over 1% of the nominal AFTA run-time system iteration rate. Token acquisition latency for synchronous data will be no larger than 8.0 ms; a smaller value may be established at ring initialization time.

The token passing media access control guarantees access to the network by each station within a predetermined time period. Each station uses a token holding timer (THT) to limit the amount of time the station transmits on the network. A properly designed station will release the token when the THT expires. A station which does not relinquish the token after the THT expires is faulty.

The token passing protocol in FDDI is also designed to handle synchronous and asynchronous bandwidth. Synchronous bandwidth is allocated statically for each station and is guaranteed. Allocation of synchronous bandwidth is done in a manner to ensure that the total synchronous bandwidth does not exceed the maximum practical bandwidth capacity of the physical link. After all stations have transmitted synchronous data, any bandwidth left over is available for asynchronous bandwidth.

An FDDI network is constructed using either a single ring or a dual, counter-rotating ring configuration. The dual ring design provides some degree of fault-tolerance to the network. One ring in the dual ring configuration is deemed the primary ring and the other ring is the secondary ring. The primary ring is used unless a fault is detected, at which point all stations switch over to the secondary ring. An additional fault on the secondary ring can sometimes be tolerated by connecting segments of the two rings into a new configuration.

Each network interface unit on an FDDI network is either a single attach or a dual attach station. Single attach stations are only connected to the primary ring, whereas dual attach stations connect to both rings. Dual attach stations only listen on one ring at a time. Since dual attach stations connect to both rings, a single malicious fault in a network interface unit can interrupt communications on both rings simultaneously. Thus, the dual ring FDDI configuration is not sufficient alone to provide Byzantine resilience.

An FTDB implementation utilizing FDDI requires at least two distinct FDDI networks for Byzantine resilience. Each network, henceforth referred to as a media layer, is used to transmit a packet copy between stations. Stations are connected to the dual media layers in a manner that prevents any single fault within a fault containment region from disrupting more than one media layer. Each media layer is either a single or a dual FDDI ring.

An FTDB implementation using dual, counter-rotating FDDI rings uses the secondary rings to reconfigure around diagnosed passive faults. Note that even an implementation using single FDDI rings is Byzantine resilient.

A diagram of the FTDB architecture is shown in Figure 6-8. The FTDB supports stations of simplex and fault-masking (triplex or quadruplex) redundancy levels. The FTDB protocols guarantee agreement on data transmitted from a simplex to a fault-masking group. Validity is guaranteed if the simplex source is functional. The FTDB also guarantees agreement and validity on data transmitted between fault-masking groups.

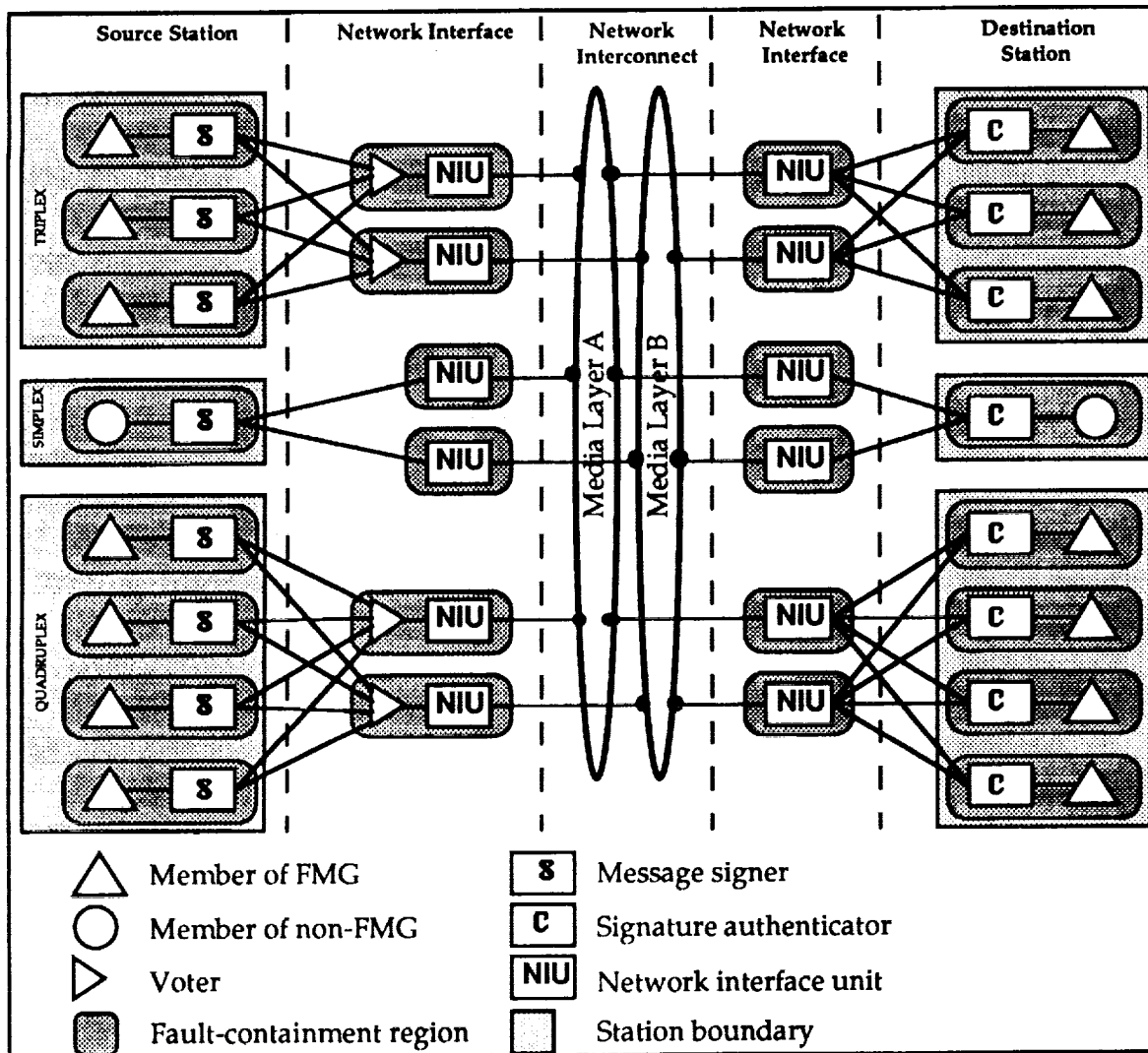


Figure 6-8. FTDB Architecture

The FTDB is built around two FDDI media layers to provide reliable communication between network stations. Each media layer is physically and electrically isolated from the other layer and from all network station members; thus a single failure in the system will disrupt at most one media layer and will not disrupt any station member. A media layer may be either a single or a dual FDDI ring.

All stations are connected to the network using one of the interface architectures shown in Figure 6-8. All stations must provide a network interface unit (NIU) to each of the two redundant media layers. Each NIU must reside in a separate fault-containment region to ensure Byzantine resilience.

The steps taken by a message as it is transferred through the FTDB are described below. These steps are illustrated for a source consisting of a triplex AFTA Virtual Group sending a message to a destination consisting of an arbitrary triplex processing site. The AFTA Virtual Groups view the interface to the FTDB as simply another type of I/O Controller, and use the I/O message exchange primitives enumerated in Section 3. To illustrate the linkage between the AFTA and the FTDB, Figure 6-9 is a redrawing of Figure 3.43, showing how triply redundant VG T1 simultaneously writes voted output data to multiple IOCs, which in this case are the Signer/Checker (S/C) components of the FTDB.

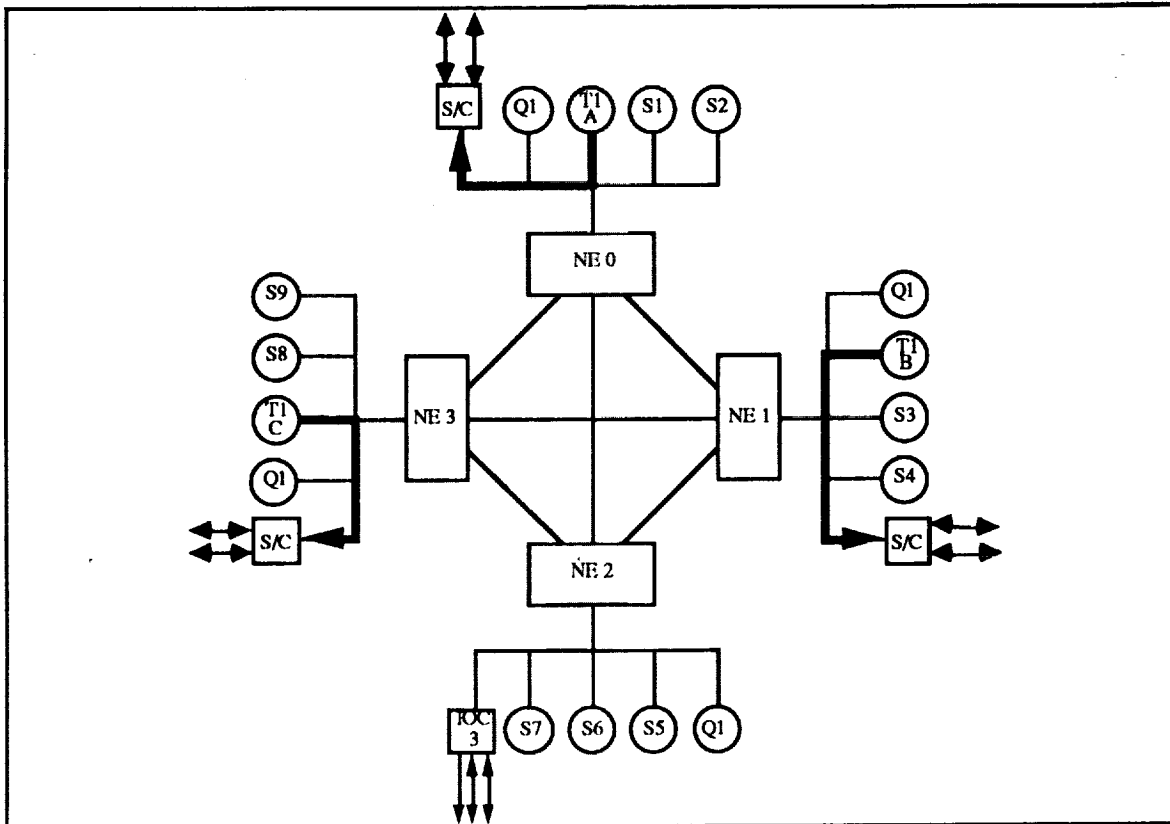


Figure 6-9. Triply Redundant VG T1 Simultaneously Writes Output Data to Signer/Checker Components of Fault Tolerant Data Bus

Step 1, Figure 6-10. Data transmitted from the station to the network passes through a message signer. The signer is in the same FCR as the data source. The signer attaches a sequence number and an authentication signature to the packet.

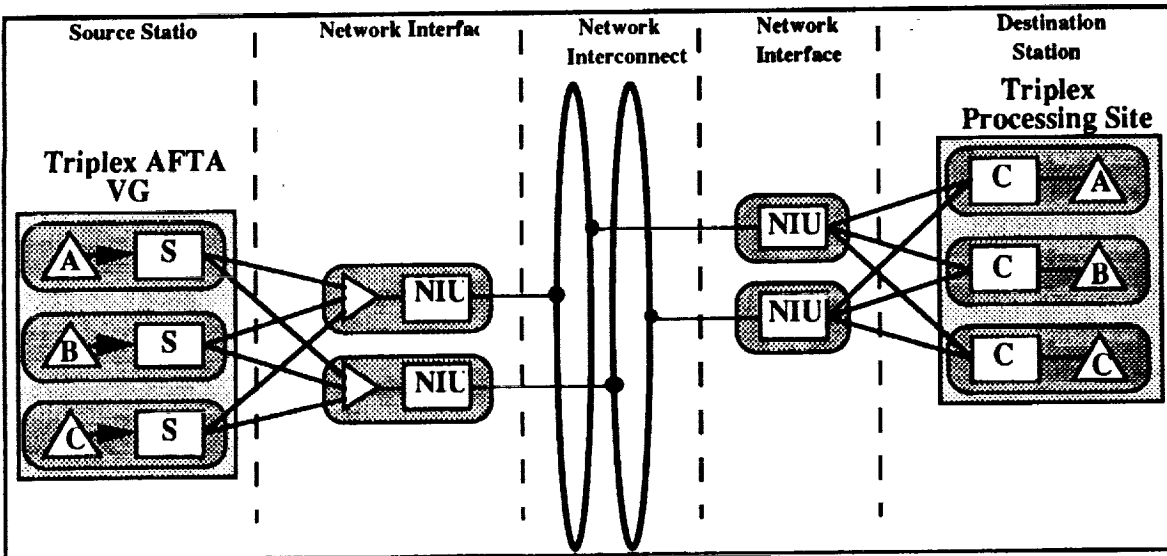


Figure 6-10. Step 1 of FTDB Message Transfer

Step 2, Figure 6-11. After the authentication information is attached to the packet, the packet is transmitted to the network interface units. In the case of an FMG, the redundant copies of the packet are transmitted to the FTDB interfaces, each of which votes the three copies to produce two redundant, signed packets.

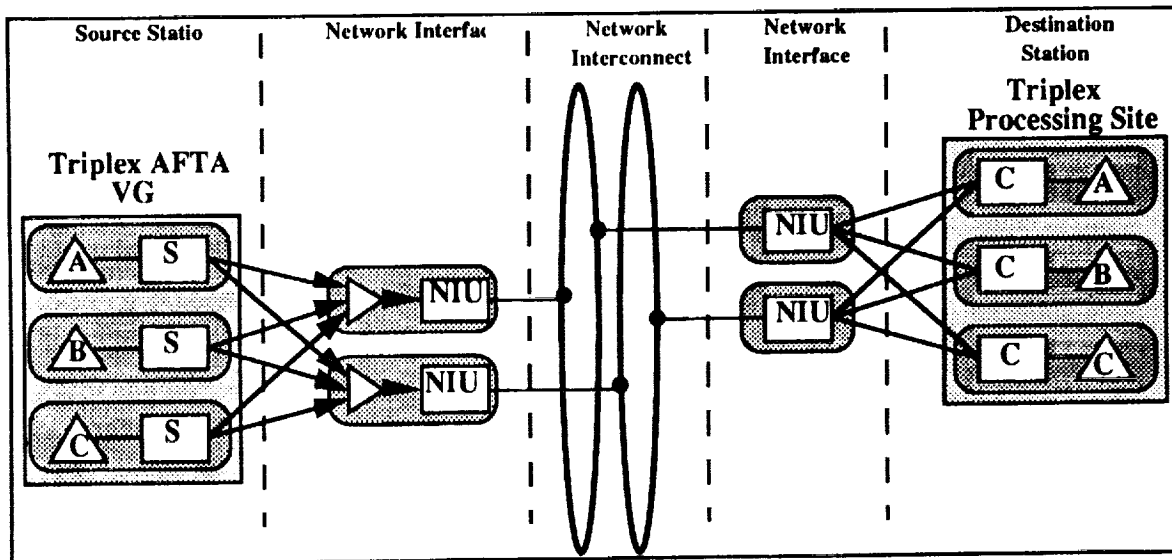


Figure 6-11. Step 2 of FTDB Message Transfer

Step 3, Figure 6-12. Each packet is then transmitted over the dual FTDB media layers to the appropriate receiver station.

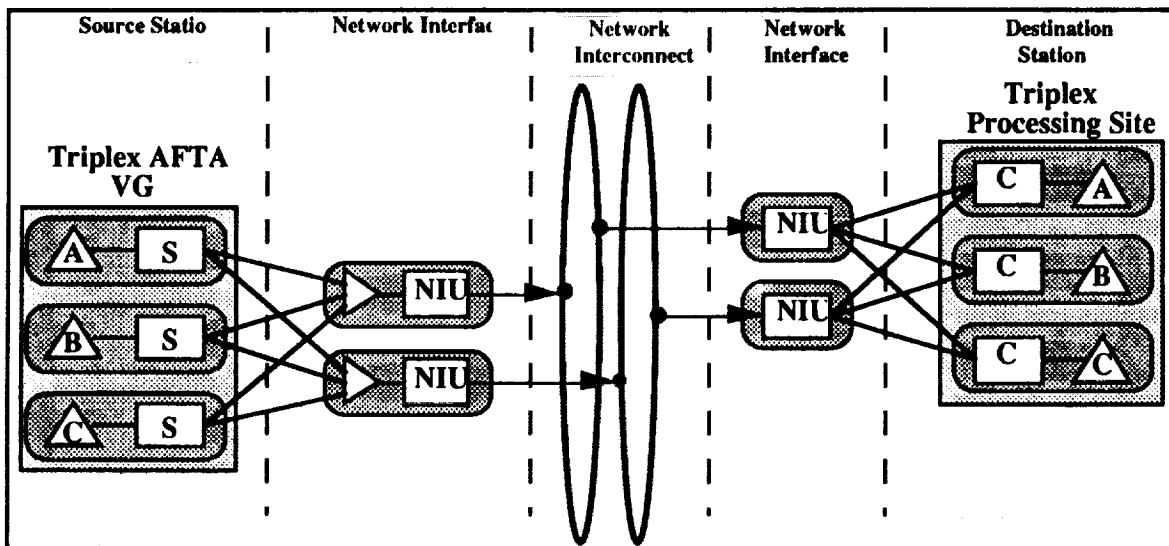


Figure 6-12. Step 3 of FTDB Message Transfer

Step 4, Figure 6-13. The receiving FTDB NIUs transmit the received packet copies to the signature authenticator stage of the receiving station. The authenticator stage nominally receives two copies of each packet, one from each media layer. One copy is guaranteed to be correct in the presence of any single fault in the network.

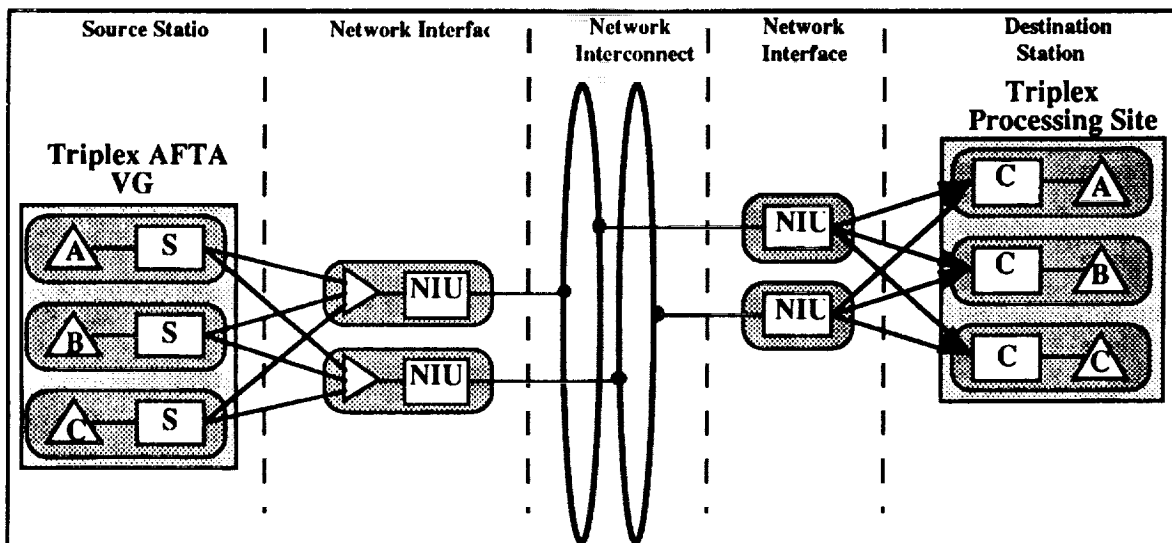


Figure 6-13. Step 4 of FTDB Message Transfer

Step 5, Figure 6-14. The authentication stage checks the sequence number and the signature on the packet to make sure the packet is valid. Both the sequence number test and the signature authentication test must succeed for the packet to be considered valid. If both packet copies available at a given checker pass both tests, either copy may be selected as

valid by the receiving member of the destination triplex processing site. If neither copy passes both tests, the packets are discarded and the fault information is recorded in the network diagnostic log. Subsequently, the received packets and fault information may be exchanged using one of the standard AFTA I/O exchange primitives enumerated in Section 3. The specific exchange primitive used will depend on the redundancy level of the recipient VG and the number of FTDB interfaces that VG possesses.

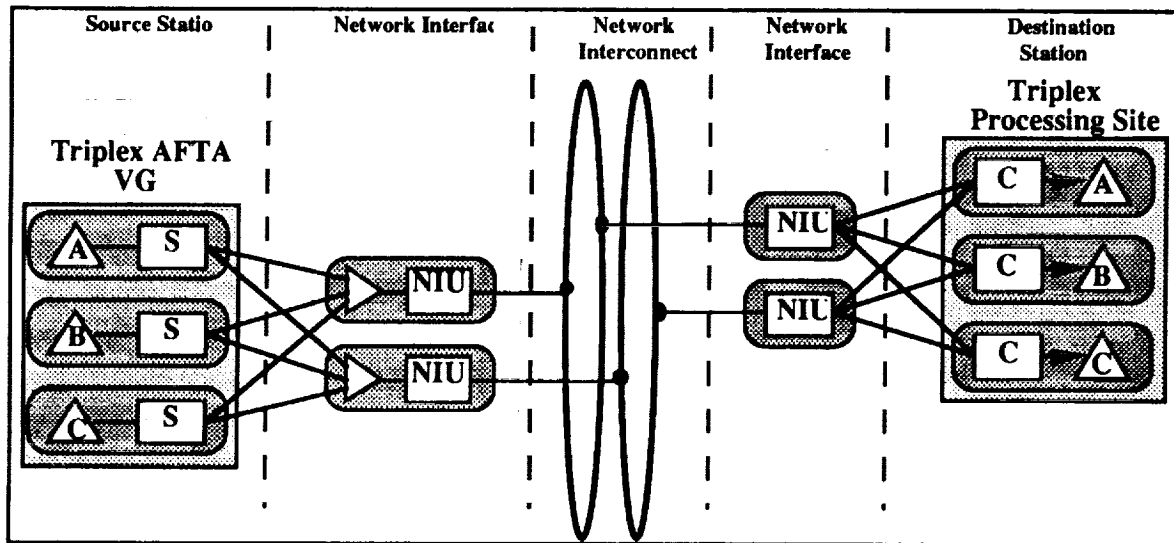


Figure 6-14. Step 5 of FTDB Message Transfer

6.5.1. Physical Layer

The physical layer of the FTDB is based on the specifications for the Fiber Distributed Data Interface, or FDDI. The physical layer specification in the FTDB is divided into two major segments. The physical layer protocol defines the data and control signaling and clock recovery. The physical layer medium dependent describes the actual electrical and optical hardware used to implement the inter-station communication link.

6.5.1.1. Physical Layer Protocol

The physical layer protocol (PHY) for FTDB is described in [ANSI148]. The data is encoded using a 4B/5B code to maintain a DC balance on the output waveform. The code also ensures that the serial data stream will contain no more than three adjacent zero symbols. This property assists the clock recovery circuitry by providing enough transitions to derive the clock from the incoming data stream. The stream of 5 bit codes is converted into an NRZI serial data stream for transmission over the serial medium.

The use of the 4B/5B code makes better use of the media bandwidth than Manchester encoding. For example, the raw bandwidth of the FDDI medium is 125 Mbaud. Using 4B/5B encoding, a data rate of 100Mbits/sec is obtained. Using Manchester encoding, only 62.5Mbits/sec would be available. The 4B/5B coding requires a more sophisticated clock recovery circuit and more accurate oscillators than Manchester encoding. However, oscillators satisfying the 50 ppm specification for FDDI are widely available, as are monolithic integrated circuits to perform the clock recovery [AMD89a].

6.5.1.2. Physical Layer Medium Dependent

The FDDI physical layer medium dependent (PMD) standard [ANSI166] defines the physical medium to be used for the data communication channel. The standard includes specifications for fiber-optic type, fiber-optic diameter, light wavelength, transmitter type, receiver type, and connector dimensions. These specifications are either necessary to achieve the 125MHz signalling frequency or to ensure compatibility between stations on an FDDI network.

Other medium dependent specifications can be used to replace the FDDI PMD, provided that the replacement specification is compatible with the rest of the FDDI specification. Two examples of alternate medium dependent specifications are the Militarized Fiber-Optic Transmission System specified by FDDN [Coh88] and the SAFENET II media dependent layer [MIL-HDBK-0036]. Each of these specifications defines militarized components not considered by the FDDI PMD specification. However, both are compatible with the remainder of the FDDI specification.

The modularity of the FDDI PMD permits the substitution of different physical medium dependent layers on a per-network basis. Thus, a militarized FTDB network can be constructed by simply replacing electrical components with their military equivalents, and replacing the PMD layer with a militarized medium dependent layer, such as one of the two examples presented above.

6.5.2. Data Link Layer

The data link layer of the FTDB is based on the FDDI data link layer standard and the IEEE standard for logical link control. The FDDI media access control arbitrates access to the physical network. The FDDI station management protocol provides a set of primitives for maintaining the processes in the physical and data link layers. The IEEE logical link

control (LLC) maintains the link between the physical layer and the network layer protocols, and provides peer-to-peer communication with other LLC entities on the FTDB.

6.5.2.1. Media Access Control

The media access control (MAC) for the FTDB is defined in the FDDI standard [ANSI139]. The FDDI MAC is based on a token passing protocol. Token passing exhibits many characteristics desirable for real-time systems, including guaranteed deterministic, low latency data transmission and the ability to schedule synchronous data exchanges.

Deterministic token rotation is guaranteed using a token rotation timer (TRT). The TRT is initialized during a bidding process. Each station on the network broadcasts a desired maximum token rotation time. If the station receives a request for a TRT less than what the station requested, the station drops out of the bidding. If the received TRT request is greater than that the station requested, the station ignores the request and continues the bidding process. The last station in the bidding selects the local TRT request and broadcasts it to all other stations as the target token rotation time (TTRT). The bidding process ensures that the shortest TRT request is used for the TTRT.

The TTRT is used during normal data transmission to prevent a station from monopolizing the network. A token that arrives before the TTRT is an early token and can be used to transmit either synchronous or asynchronous data. A token that arrives after the TTRT is a late token and can be used only to transmit synchronous data. Synchronous bandwidth is allocated such that all synchronous data is guaranteed to fit within one TTRT. Each station should normally have an opportunity to transmit every token rotation time. Access to the network is guaranteed within two target token rotation times.

The MAC specification also defines the station addressing protocol. FDDI station addresses are categorized by three characteristics as outlined below.

- Physical, logical, or broadcast. Only one station listens on a physical address. Multiple stations may listen on a logical address. A station may listen on more than one logical address. All stations listen on the broadcast address.
- Universal or local administration. Universally administered addresses are assigned by a single authority and are guaranteed to be unique throughout the world. Locally administered addresses are assigned by the manager of the local network. The local manager is responsible for preventing local address conflicts.
- Length. FDDI addresses are either 16 bits (short address) or 48 bits (long address) in length. Only long addresses can be universally administered.

6.5.2.2. Station Management

The station management of an FTDB station is defined by the station management (SMT) standard for FDDI [X3T95]. The SMT controls various processes in an FTDB node, including station insertion and removal, initialization, fault detection, isolation and recovery, ring bandwidth allocation and scheduling, and configuration management. Although included with the data link layer in this discussion, SMT actually controls the local PMD and PHY entities in the physical layer as well as the MAC entity in the data link layer.

6.5.2.3. Logical Link Control

The FTDB logical link control (LLC) protocol conforms to the LLC defined in [IEEE8022] and [IEEE8021]. The IEEE LLC is not a part of the FDDI specification, but most FDDI implementations (including SAFENET II installations) use this LLC by convention. Conformance to the IEEE standard ensures compatibility with other FDDI stations using the same FDDI media.

The LLC defines service access points (SAPs) which specify the location to which the LLC delivers incoming packets. The destination SAP (DSAP) usually indicates a network layer protocol stack. Most SAPs are reserved by IEEE for use by public protocols. However, the LLC reserves one SAP for use in a protocol extension, known as the sub-network access protocol (SNAP), for private protocols. Since the Byzantine resilient network protocol of the FTDB is considered a private protocol, the FTDB LLC uses the SNAP extension to distinguish BRNP packets from other types of packets.

The LLC defines both datagram (Type 1) and connection-oriented (Type 2) communication between SAPs. Implementation of Type 1 is required, whereas Type 2 functionality is optional. The FTDB only requires Type 1 capabilities, since the BRNP is designed around datagram protocols. However, the FTDB may optionally include Type 2 capabilities if a protocol stack that requires Type 2 capabilities is developed for the FTDB.

6.5.3. Network Layer

The network layer protocols are responsible for fulfilling station requests for message transmission, address resolution, message authentication, and message delivery to stations.

6.5.3.1. Byzantine Resilient Network Protocol (BRNP)

The Byzantine resilient network protocol implements the Byzantine Resilient Virtual Circuit (BRVC) model [Har87]. The BRVC model guarantees delivery of all messages transmitted by BRNP. Most existing network layer protocols, IP, for instance, are best-effort systems that make no guarantees about message delivery; the transport layer protocols are responsible for providing reliable communication through retry mechanisms. However, retry mechanisms can be fooled by Byzantine faults [Ber87]. Therefore, true Byzantine resilience can not be implemented unless the underlying network layer protocol supports Byzantine resilience through redundancy and/or message authentication.

BRNP supports communication between stations of varying redundancy levels. The three redundancy levels of the AFTA (simplex, triplex, and quadruplex) are currently supported. BRNP guarantees agreement on data transmitted from a simplex to a fault-masking group. Validity is guaranteed if the simplex source is functional. The FTDB also guarantees validity on data transmitted between fault-masking groups.

BRNP supports synchronous and asynchronous bandwidth. Synchronous data will always pre-empt asynchronous data in the output queue. The token passing media access control of FDDI supports this model well. This data model blends well with the AFTA run-time system model of synchronous (rate-group) and asynchronous (background) tasks.

Data link and physical layers connected to BRNP must provide at least two connections to the media layers they represent. These two connections must be ports into mutually exclusive paths to all other stations that support BRNP. The mutually exclusive paths are required so that BRNP can transmit two packet copies that traverse the network with no interconnecting link or node in common. If the packet copies were to pass through the same link or node, the FTDB would be susceptible to single point failures.

The transmitting BRNP entity receives message transmit requests from the transport layer protocols in the station. The transmitting entity communicates with the peer BRNP entity on the receiving station over the redundant FTDB communication links. The message received from the transport layer is inserted into a BRNP packet with a sequence number and a signature from the authentication protocol. The BRNP packet is then delivered to the LLC entity in the station, which transmits a copy of the packet over each of the dual FTDB media layers to the receiving station.

The receiving BRNP entity is responsible for resolving the multiple packet copies arriving over the two media layers into a single copy to be delivered to the transport layer. The two media layers are not bitwise or token synchronized. However, the maximum latency of a packet on the physical layer is bounded by the token rotation timer. This characteristic is used by the receiving BRNP entity to maintain functional synchronization of the two media layers.

When BRNP receives a packet, the sequence number is checked to see if the other copy of the packet has already arrived. If so, the packet is treated as the second copy, otherwise the packet is treated as the first copy. The packet is processed by ATP to determine the validity of the sequence number and the signature. The results of the validation test are ANDed together to create a packet status bit (PSB). A fault masking station performs a source congruency on the PSB of each member to generate a packet status vector (PSV).

The PSV for the first packet copy is treated using the following protocol:

- If no members receive a valid packet, the packet is discarded.
- If a minority of members receive a valid packet, a timeout equal to 2 times the TTRT is started. If the second packet copy does not arrive before the timeout expires, the packet is discarded.
- If a majority of members receive a valid packet, a timeout equal to 2 times the TTRT is started. If the second packet copy does not arrive before the timeout expires, the packet is delivered and the sequence number is incremented.
- If a unanimity of members receive a valid packet, the packet is delivered and the sequence number is incremented.

The PSV for the second packet copy is processed using the following protocol:

- If a minority or no members receive a valid packet, both copies of the packet are discarded.
- If a majority or a unanimity of members receive a valid packet, the second copy of the packet is delivered and the sequence number is incremented.

The protocol outlined above ensures the earliest delivery of valid packets to their destination while still guaranteeing correct behavior in the presence of a single Byzantine fault. One interesting characteristic is that if the first packet is valid on all receiving station members, that copy will be delivered immediately and the second packet copy will be discarded as invalid since the sequence number is incremented on delivery. Another characteristic is that the network peer-to-peer latency is no worse than for a simplex physical layer, whether or not faults are present. In a fault-free system, the fastest physical layer will deliver the first copy, which will be immediately delivered to the destination. If

the first packet is lost or corrupted, the second packet copy will still arrive within the predetermined latency ($2 \times T_{TRT}$).

6.5.3.2. Authentication Protocol (ATP)

The Byzantine resilient network protocol uses the authentication protocol (ATP) to sign outgoing packets and to test the authenticity of incoming packets. The transmitting ATP entity attaches two pieces of information to each outgoing packet for use by the receiving ATP entity: a sequence number and a signature. These two items are used by ATP and BRNP to determine if a given packet was sourced by the appropriate station, and to select a valid packet from multiple packet copies.

The sequence number defines the sequence of packets leaving a station destined for a specific address. Two sequence number tables are kept by each station, one for transmit and one for receive. The transmit sequence number table contains a separate sequence number for each physical, logical, and broadcast address to which the local station transmits. When a sequence number is requested by BRNP for a destination address, the ATP returns the current sequence number in the table and automatically increments it in preparation for the next packet.

A corresponding sequence number table is kept with sequence numbers for each physical, logical, and broadcast address the station listens on. When a packet is received from a remote station, the ATP checks to see if the sequence number on the packet is the number in the receive table. If the sequence number is correct, the packet passes the sequence number test. The values in the receive sequence number table are not incremented automatically on success of the sequence number test. BRNP increments the receive sequence numbers only on delivery of a packet to a destination.

Signatures on outgoing packets are calculated by applying a private key function to the data contained in the packet, including the sequence number. Each station has a different private key function, thus no station can impersonate another station by using another station's key. The same private key function is used regardless of whether the packet is addressed to a physical, logical, or broadcast address.

The ATP validates the signature by applying a public key function to the packet data and to the signature. If the results match, the packet passes the signature authentication test. Each private key function has a different corresponding public key function, so each station

must keep a table of the public key functions for each remote station from which the local station expects to receive a packet.

The installation of a new station on the FTDB network requires synchronization of the sequence number tables and distribution of the public key for the new station.

6.5.3.3. Address Resolution Protocol (ARP)

The address resolution protocol (ARP) maps network addresses to station physical, logical, or broadcast addresses. The ARP is used by BRNP to determine the correct physical address to attach to each packet destined for another station. The assignment of network addresses is static in a particular FTDB implementation, thus the ARP is simple and fast.

Network addressing in BRNP is designed to allow a station to be addressed as a single unit, regardless of the station's redundancy level. This concept is similar to the virtual group identifier (VID) numbers used in the AFTA. Consequently, a single network address may map to several physical addresses.

6.5.4. Transport Layer

The transport layer protocols provide convenient application user interfaces for different data communication models. Each transport layer protocol communicates through BRNP. The protocols handle all redundancy management and fault-masking issues. The user programming model for each protocol is a virtual simplex unidirectional or bidirectional data port. All protocols are supported on sites of simplex, triplex, or quadruplex redundancy level. In the AFTA, the transport layer protocols are implemented in the Ada run-time system as a part of the I/O systems services. However, the protocols are not tied to any particular programming language, development environment, or operating system.

Normally, the transport layer is used to build reliable message delivery on top of an unreliable network layer. For example, the transmission control protocol (TCP) uses a retry mechanism to ensure message delivery using the best-effort internet protocol (IP). However, in the FTDB, the network protocol itself is reliable, so the transport protocols do not need to implement retry or other mechanisms for reliable message delivery.

Most of the transport layer protocols described below use sockets to discriminate between multiple application tasks using the same protocol. The socket model is similar to that employed by the TCP/IP protocol suite [Com91].

6.5.4.1. Periodic Datagram Protocol (PDP)

The periodic datagram protocol (PDP) uses a connectionless socket to periodically transmit or receive data using synchronous bandwidth. An example of a possible application for PDP is for an intelligent sensor computer that periodically transmits a sensor reading to one or more controller tasks. The protocol provides a method for synchronizing sender and receiver. Since the protocol is based on reliable datagrams, the sender does not care if the receiver is present or not; data delivery is guaranteed if the receiver is present. Data received on a PDP socket also contains a timestamp, so the user task can determine the relative age of the data. The protocol provides an exception mechanism for the receiver if an expected datagram doesn't arrive within the expected timeframe.

6.5.4.2. Services Transaction Protocol (STP)

The services transaction protocol (STP) implements a point-to-point transaction model for data transfer. A typical transaction-type situation is a server/client paradigm. The client sends a request for a transaction to the server and waits for a response. The server completes the transaction and returns the result to the client. The client blocks until the server responds. The protocol provides a services registration mechanism, so that a server does not need to always reside at the same station. In fact, a server could be moved if the station on which a server resides becomes faulty.

6.5.4.3. Asynchronous Datagram Protocol (ADP)

The asynchronous datagram protocol (ADP) uses asynchronous bandwidth and a connectionless socket to transmit data. ADP sockets are non-blocking, as a response is not expected within a short timeframe. Reception of an ADP datagram is treated as an exception. A task can either use an exception mechanism or polling to determine if an ADP packet has arrived. ADP supports point-to-point and multicast communication. The ADP programming model is very similar to the `send_msg` and `get_msg` primitives of the intra-cluster communication system in the AFTA Ada run-time system.

6.5.4.4. Network Data Stream Protocol (NDSP)

The network data stream protocol (NDSP) provides a virtual circuit connection between two sockets. The operation is similar to that provided by TCP. Asynchronous or synchronous bandwidth can be used. An attempt is made to allocate synchronous bandwidth, if requested, when an NDSP socket is opened. If the synchronous bandwidth is not available, asynchronous bandwidth is used until the requested synchronous bandwidth becomes available. If there is no outgoing data in the NDSP output buffer, the synchronous bandwidth is wasted, so synchronous NDSP sockets should be used with discretion.

6.5.4.5. Network Diagnostic Protocol (NDP)

The network diagnostic protocol (NDP) is used to perform diagnostic testing of the network. An NDP datagram can be pre-routed by the sender and can be transmitted through a loop back to the sender. NDP can also request nested authentication, so that each station that supports BRNP will concatenate its signature to the NDP datagram, providing a mechanism for testing each node on a network.

The NDP includes a diagnostic log which records recently observed authentication errors by the local authentication protocol. The network FDIR task uses the diagnostic log to attempt to diagnose faults in the system. Also, the diagnostic log can be copied to a permanent database for use in system maintenance.

The NDP also provides a mechanism for reconfiguring around diagnosed faults in the physical layer if the physical layer supports reconfiguration. In the FDDI implementation of the FTDB, the NDP interfaces to the station management (SMT) entities for reconfiguration of a media layer.

Since network diagnosis and reconfiguration is dependent on the physical layer, NDP is dependent on a particular physical layer implementation.

6.5.4.6. Echo Protocol (EP)

The echo protocol (EP) simply returns an echo response whenever an echo request is received. The echo protocol provides a simple and convenient method for determining the status of a network station.

6.5.4.7. Time Management Protocol (TMP)

The time management protocol (TMP) maintains a global time value for all TMP subscribers. The source of the global time value can be either a single external source, such as an accurate time reference, or a distributed time agreement algorithm based on multiple, less accurate time references. The TMP is used by the PDP to determine the age of periodic data.

6.6. FTDB Development Plan

This section presents a proposal for development of a fault-tolerant data bus brassboard system. The FTDB brassboard can be used to interconnect redundant (C2, AFTA) and non-redundant (Silicon Graphics, MT-1) computing sites. The development plan is segmented into subtasks, with each subtask emphasizing a different area of development.

6.6.1. Developmental and Non-developmental Items

This section details the hardware items to be developed or acquired as part of the proposed FTDB development plan.

The authenticator module (ATM) is a developmental item. The ATM implements the authentication protocol (ATP) described above. The ATMs reside in the same FCRs as the AFTA, functioning as a redundant I/O device in the AFTA I/O paradigm. For an FMG, there are 3 or 4 authenticator modules, depending on the redundancy level of the FMG. The ATM signs outgoing messages, and authenticates signatures on incoming messages. The design of the ATM is based on an ATM designed under a CSDL IR&D project.

FDDI interface boards are available as non-developmental items. The boards tentatively selected for the initial FTDB brassboard are the Interphase V/FDDI 4211 Peregrine boards. These boards are available with either a single attach or dual attach interface and contain an embedded AMD 29000 Processing Element to assist in implementation of the data link protocols, especially the station management protocol. This processor can also implement parts of the network layer protocols for the FTDB.

The voting interface module (VIM) is a developmental item. The VIM takes 1, 3, or 4 copies of a signed packet from the ATM and votes them. The VIM resides in the same FCR as the FDDI interface board. The VIM is a very simple device. The design of the VIM is based on the receive and vote stages of the AFTA Network Element.

An optional developmental item is a single board interface module. The single board interface reduces the FTDB interface from 2 boards per layer (not including the ATM) to one board per layer by combining the VIM with an FDDI interface. The development of the single board interface module requires building a custom FDDI interface using a commercially available FDDI chip set such as that available from Advanced Micro Devices or National Semiconductor.

6.6.2. Proposed FTDB Brassboard Development Plan

The proposed FTDB brassboard development plan is segmented into 4 subtasks. Each subtask is considered an upgrade to the previous subtasks, so the costs of each subtask are incremental.

6.6.2.1. Subtask 1-Authentication Protocols

The emphasis of subtask 1 is to demonstrate the use of signed messages for authentication, the embedding of authentication protocols onto an existing network standard, and compatibility between BRNP and other FDDI traffic. The system developed under subtask 1 also serves as a base for additional protocol development.

The characteristics of the system developed under subtask 1 are as follows:

- 2 station system of simplex devices
- single attach FDDI
- single media layer

Tasks:

- 2 authenticator modules
- 2 voting interface modules
- 2 single attach FDDI interfaces
- 2 FCR enclosures
- data link layer protocols
- BRNP, ATP
- system integration

6.6.2.2. Subtask 2-Byzantine Resilience

The system developed under subtask 2, an upgrade to the system built under subtask 1, demonstrates the Byzantine resilience of the FTDB, support for mixed redundancy, and the reduction in hardware required for the network interface unit. One of the stations in the subtask 1 system is upgraded to a fault-masking group, and an additional FMG is connected.

The characteristics of the subtask 2 system are as follows:

- 3 stations, 2 fault-masking
- single attach FDDI
- dual media layers

Tasks:

5 authenticator modules
4 self-contained interface modules
4 FCR enclosures
system integration

6.6.2.3. Subtask 3-Network FDIR

Subtask 3 demonstrates the ability to diagnose and reconfigure around faults in the network. To demonstrate this capability, all stations must be upgraded to dual-attach FDDI, and the network diagnostic protocol and network FDIR tasks must be written and interfaced to the station management protocol. Software and hardware fault injection is used to test the functionality of NDP and the network FDIR task.

The characteristics of the subtask 3 system are as follows:

- 3 stations, 2 fault-masking
- dual attach FDDI
- dual media layers

Tasks:

6 dual attach upgrades
NDP, network FDIR, fault injection

6.6.2.4. Subtask 4-Transport Layer Protocols

The purpose of subtask 4 is to develop transport layer protocols for use by user application tasks on the AFTA or other FTDB subscriber. These protocols are intended to make the FTDB useful in a real-time system. The subtask 4 system demonstrates the applicability of these protocols for real-time applications. At the conclusion of subtask 4, all developmental work for the brassboard FTDB is complete.

Tasks:

transport layer protocols

6.6.3. FTDB Brassboard Development Schedule

Figure 6-15 describes the development of the FTDB brassboard. The development schedule is intended to correspond to the development of the AFTA FTTP. Delivery of the FTDB brassboard is targeted for sometime in March, 1993, near the projected delivery date for the rest of the AFTA brassboard design.

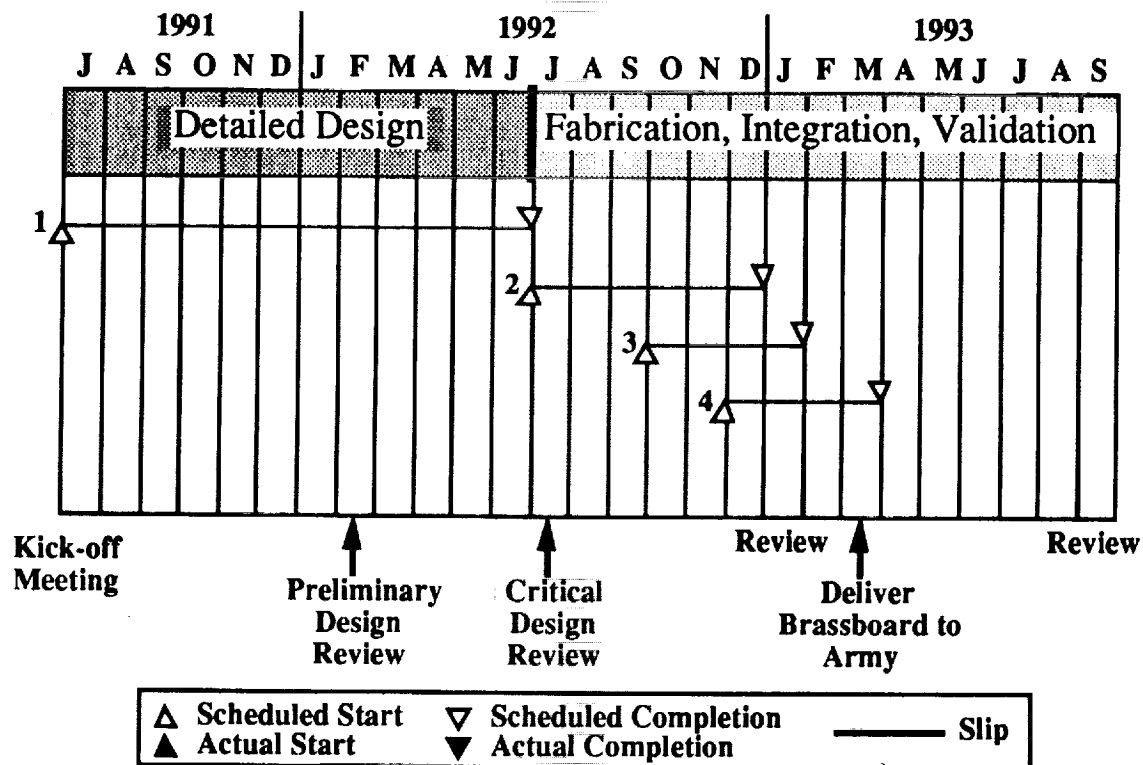


Figure 6-15. FTDB Brassboard Development Schedule

This page intentionally left blank.

7. Testability and Maintainability

AFTA is designed to be testable for hardware faults at all stages of its lifetime. As a fault tolerant computing system, it actively tests itself during operational modes in order to maintain its high reliability. During mission critical operations it is imperative that faulty components be identified and expunged from the system to prevent the possibility of a system failure should a second uncovered fault occur. However, because no computing system is operational 100% of the time and because all digital computing systems require maintenance, the testing capabilities of the AFTA will also encompass maintenance modes of operation as well. Consequently, the system test activities will address all aspects of determining hardware faults – at the maintenance depot, upon command by an operator, at power on, and in a mission critical environment.

7.1. Level of testing

The AFTA consists of numerous, individually testable components. Testing of the AFTA will exercise these components as comprehensively as possible. The components addressed by the test suites are the processors, Network Elements, I/O controllers, power conditioners, mass memory devices, and VME buses.

There are essentially 2 levels of testing – component self tests and system tests. The component self tests are intended to isolate faults in the functional components of a line replaceable module with the emphasis on isolating the fault to a chip-level component. This goal can be achieved using on-board diagnostic mechanisms or functionally equivalent tests. On the other hand, the system tests are designed not only to exercise the numerous components in a cohesive manner but also to perform these tests while performing mission critical operations.

7.1.1. Component self tests

The component self tests are diagnostic tests which exercise the various hardware components of the AFTA. Each line replaceable module (LRM) in the AFTA will have a suite of tests which exhaustively tests each functional component of the LRM. Whenever available, these tests will be supplied by the manufacturer of the LRM. The testable components on the AFTA are the processors, Network Elements, I/O controllers, VME bus, mass memory and power conditioners.

7.1.2. System tests

The component self tests exercise the functionality of the individual line replaceable modules. Conversely, the system tests exercise functions requiring multiple components operating in tandem to effectively test the system. Because the AFTA is designed as a fault tolerant system, fault detection mechanisms are built into the specially designed interconnection network and are exercised at every message exchange to provide high coverage of faults with low fault latency. The goal of the system tests is to test the AFTA as an operating entity exercising these fault tolerant mechanisms.

Fault tolerance in the AFTA is implemented using hardware redundancy. A specially designed set of Network Elements operate in tight synchrony to implement fault tolerant message exchanges among processors grouped into redundant virtual groups. The constituent processors in a virtual group communicate with the members of its virtual group and with other virtual groups by synchronously sending messages via the Network Elements. The Network Elements perform fault tolerant specific operations on messages and deliver voted messages to all members of the destination virtual group. The voting process generates a consistent voted copy of the message as well as error syndrome data which are appended to the delivered message. This error syndrome information can be used to identify faulty components.

7.2. Test Modes

AFTA testing activities shall operate in 4 distinct test modes defined by both the operational as well as the physical environment. In addition, these modes will dictate the operator interface. These testing modes are: depot test, maintenance built-in test (M-BIT), initiated built-in test (I-BIT) and continuous built-in test (C-BIT).

The depot test mode comprises a suite of tests available to the test technician or automatic test equipment (ATE) for testing the components of the AFTA at a maintenance repair facility. Specifically, the test suite consists of sets of diagnostic level tests for the processors, I/O controllers, the Network Element, VME bus, mass memory, and power conditioners. These depot tests execute outside of the constraints of a real-time environment with the emphasis on the isolation of chip level faults in these components.

The M-BIT mode is essentially flight-line maintenance which is initiated upon command by the flight or maintenance crew. Because this test mode is a maintenance mode with the emphasis on detecting and isolating faults, mission critical operations will not be

active during this stage. The computing resources are devoted entirely to this maintenance activity. Consequently, the suite of tests can be very extensive in testing the functionality of each line replaceable module of the AFTA. In addition, the test suite will include tests of the functionality of the LRM interfaces, particularly the buses. As an exhaustive set of tests, this test suite will probably require on the order of minutes to complete; however, abort mechanisms will be provided to terminate the test activity prematurely.

When power is applied to the all components of the AFTA hardware, the I-BIT mode shall be initiated. The objective of testing during this period is to identify faulty components in a non-mission critical stage to obviate their sudden exposure at a time when the recovery options are very limited. As a result of the evaluation of the series of I-BIT tests, the faulty and non-faulty components will be identified and the initial system configuration will be established concordant with the mission reliability requirements and the availability of non-faulty components. During a relatively short period (seconds), the system initializes and tests itself. However, because of the time constraint a broad spectrum of tests will be executed to test the basic functionality of all LRMs rather than extensively testing of only a couple of LRMs. Testing at this stage of activity with a comprehensive suite of tests ensures that the system reliability is as high as possible by determining those LRMs which are faulty and excluding them from the initial system configuration. Because this stage is not executing mission-critical functions, the suite of tests can be as extensive as time permits in exercising all functions of the component without regard to the maintenance of mission critical information. Furthermore, the system configuration options are far more numerous at this stage than during mission critical operation where real-time constraints are serious barriers to many reconfiguration alternatives.

The C-BIT mode will be initiated when mission critical operations are activated. During this test mode not only will the mission-critical application tasks execute within a real-time scheduling scheme but, because the system configuration consists of redundant groups, redundancy management functions such as voting will be active to ensure that failures do not disrupt correct system operation. Unlike the previous test modes, the fault detection and analysis functions are constrained within a real-time framework. In addition, these functions must not interfere with mission critical operations; data integrity must be maintained and the consumption of computing resources must be minimized.

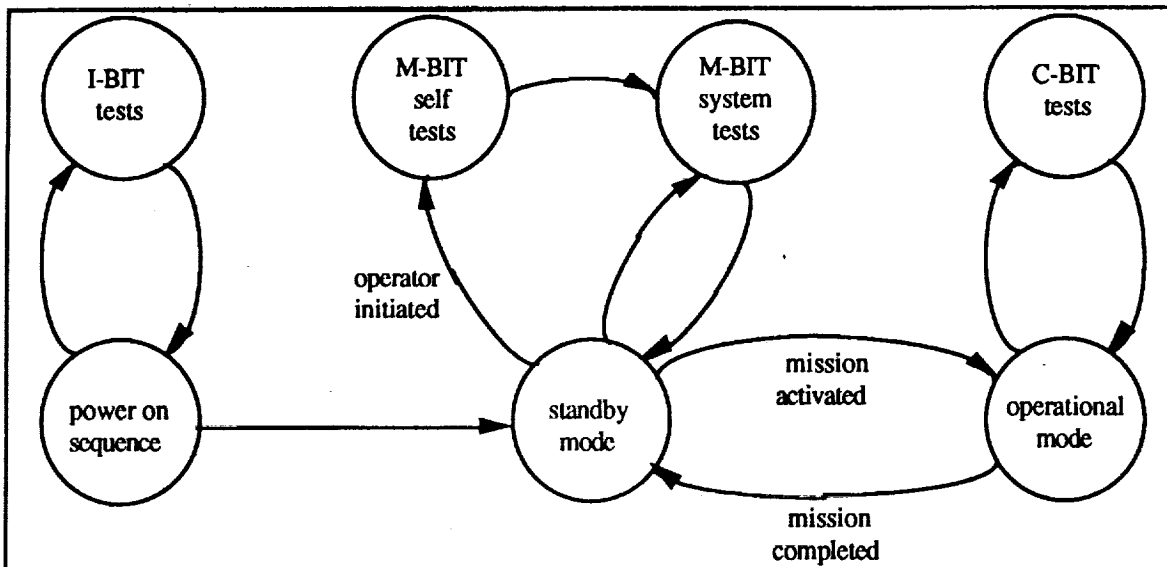


Figure 7-1. System mode and Test Mode Interactions

Figure 7-2 illustrates in greater detail the system operations from an initial power-on state and the interaction with the AFTA test modes. This sequence is described thoroughly in Section 5.

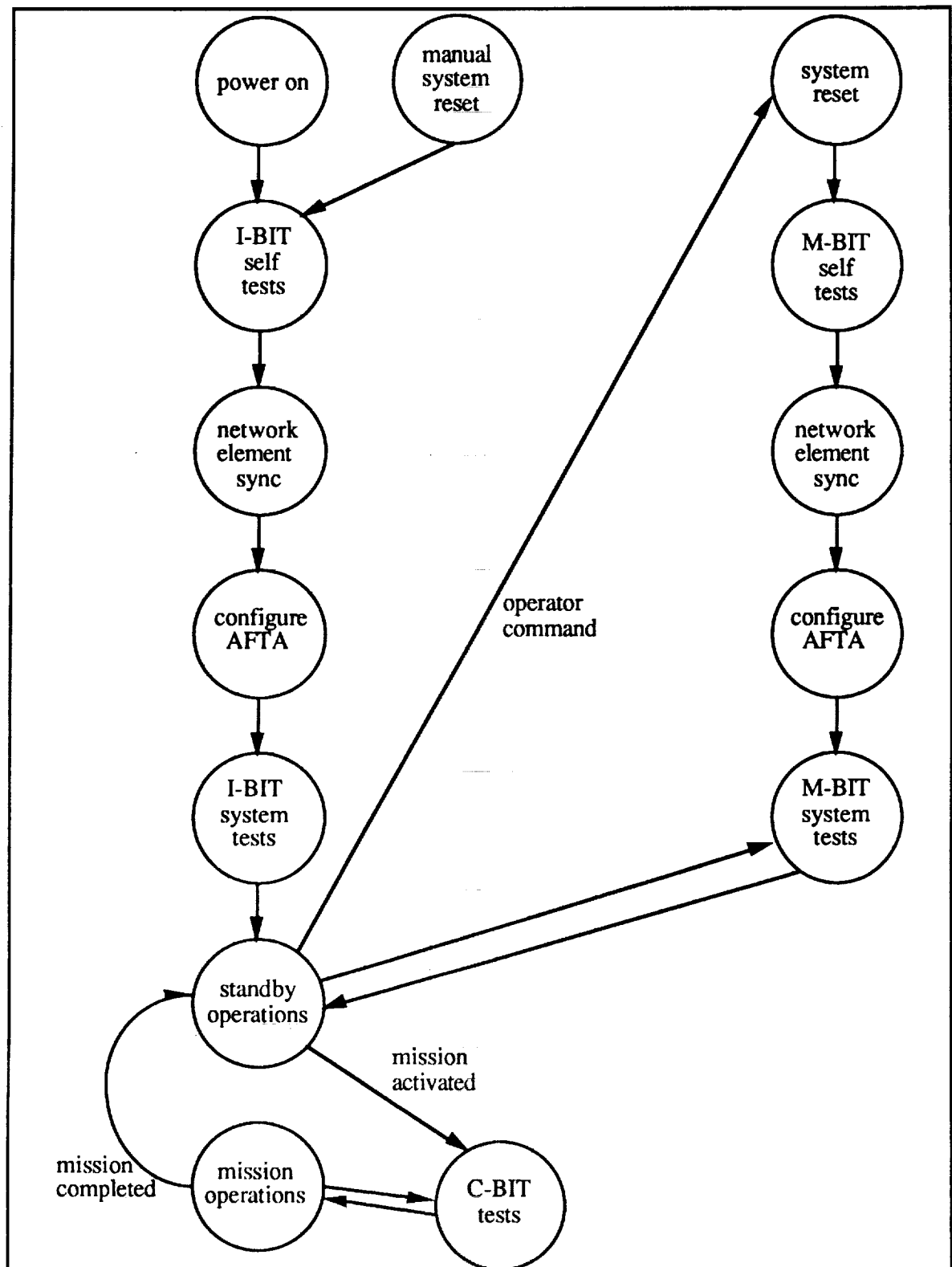


Figure 7-2. Test Mode State Diagram

7.3. Operator interface

There are 2 primary operators of the AFTA who are interested in the health of the AFTA – namely, the vehicle operator and the maintenance crew. Each has drastically differing requirements regarding the health of the digital computing system. The vehicle operator is primarily interested in discerning the relative health of the system with regards to its ability to accomplish the current mission with a sufficient measure of reliability. Specifically, the vehicle operator requires knowledge whether the mission configuration is commensurate with the requested redundancy configuration. Furthermore, he requires knowledge of the reliability of the mission configuration when it differs with the request. The AFTA operating system should provide some measure of reliability for each critical functional area – that is, navigation system, flight control, as well as the some determination of the health of redundant sensors and actuators. In addition, it is highly desirable to filter this information sufficiently such that the information presented to the vehicle operator is easily decipherable and interpretable given that the pilot may be immersed in other mission critical operator tasks.

Conversely, the maintenance crew is interested in the isolation of faults to a specific component. In fact, 2 tiers of fault diagnosis are highly desirable. Field operations maintenance requires that components be easily replaceable. Consequently, identification of faulty line replaceable modules (LRM) is important to the field operations. On the other hand, the expense of LRM replacement warrants the identification of chip level faults whenever possible. This enables LRMs to be shipped to a maintenance repair facility for diagnosis of faulty components within the LRM and replacement of those components.

7.4. FTTP C2 Network Element Tests

The FTTP C2 system is the forerunner of the AFTA system. Like the AFTA, the Network Element is the integral component of this fault tolerant computing system. While there are many differences between these systems, there are many similarities in the architecture of the Network Element.

This section describes in detail the component self tests developed for the C2 Network Element.

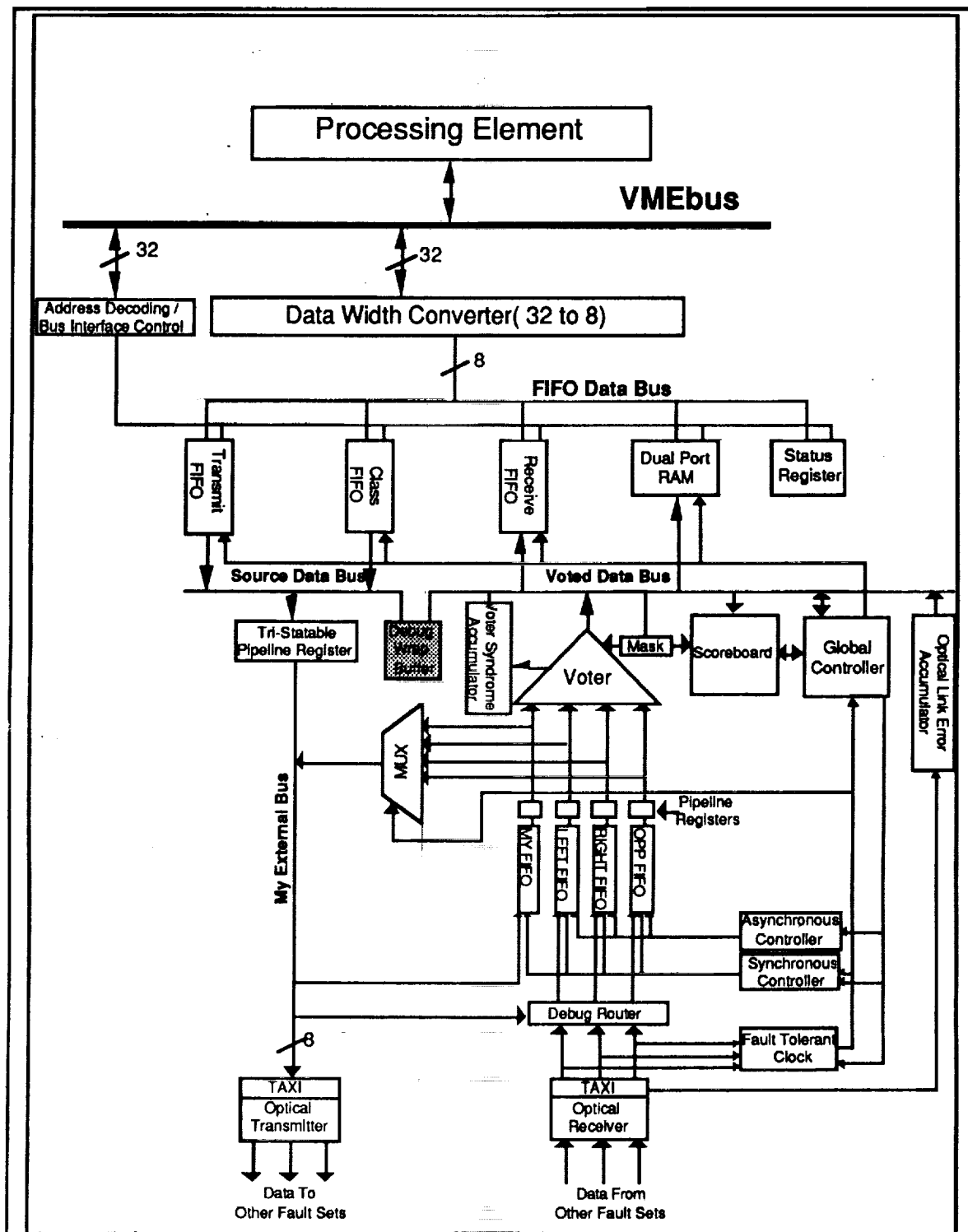


Figure 7-3. Block Diagram of FTTP C2 Network Element

7.4.1. Off-line Standalone NE Diagnostic Tests

The following tests verify the correct operation of the FFTP-C2 Network Element (NE) hardware to the extent made possible by the current design under the control of the local Processing Element(PE), an MVME-147 board. All tests are standalone in the sense that no inter-Network Element communication takes place. In fact, the execution of most of these tests would be disruptive to the synchronous operation of the aggregate NEs. Thus the test suite can only be executed by each PE running in simplex mode. The tests have been developed as a program to be downloaded to each processor and executed from RAM. However, since no static variables have been used, the code could easily be converted to a PROMable version. The PROMed version could either be called as a subroutine by any program running on the PE or as a standalone program from the 147-Bug PROMed debugger. In the first case, the subroutine returns a boolean to the calling program indicating whether or not any errors were detected. In the second case, the routine simply returns to the 147-Bug user interface. A second version of the program is provided for use from the 147-Bug user interface to allow standalone testing of the opto-electrical devices used in the fiber-optic communications. All error reporting is displayed on the VT-220 terminal attached to the PE. In addition to fully verifying the correct operation of the Network Element, the tests are intended to serve a routine maintenance function, enabling an operator to replace faulty components.

The C2 NE hardware comprises six functional blocks. They are:

- 1) The Processor-Network Element Interface
- 2) The Network Element Data Paths
- 3) The Network Element Global Controller
- 4) The Scoreboard
- 5) The Inter-Fault Set Communication Links
- 6) The Network Element Synchronization Method(Fault Tolerant Clock)

The tests in the off-line standalone test suite fully verify the correct operation of the first four functional blocks. Some of the tests also perform diagnostic analysis of errors detected during the test suite in an attempt to identify the cause of the error to as fine a level as possible. In a few cases, the IC responsible for the error can be identified. The on-card components of the fifth functional block are tested with a separate test suite since these tests can only be performed after some optical cables are connected in a testing configuration. Testing the last functional block, the Network Element Synchronization Method, requires

true synchronous NE operation. The functionality of the fault tolerant clock is minimally tested in the scoreboard tests.

7.4.2. Functional Block: Processor-Network Element Interface

Sub-block: Address Decode and Dtask Generation

Parts List: U0101, U0105, U0109, U0113, U2909

Test Description: A failure in this part of the circuitry will appear to the PE as a Bus Error. The criterion for passing the test will be that no Bus Error is detected during a read of the Status Register or the first byte in the Dual Port Memory or a write to the Class Fifo or the Transmit Fifo. Since it is not possible to isolate any of the devices, each one must be replaced in turn until the Bus Error is eliminated.

Test Sequence Number: 1. All further testing depends on the ability to correctly read and write data to this interface.

Sub-block: Reset Generation

Parts List: U0113, U0117

Test Description: Writing to the Reset Location of the DP RAM should cause the NE to reset itself. To partially verify that the reset function is operational, the reset location is written to and the necessary time delay is allowed to elapse. The status register is then read. The value stored in bits one and zero should be 1. Next a byte is written to the Class FIFO and the macro *wrap_serp_vme* is executed. The status register is again read. This time since data should be in the Receive FIFO, the value of bits one and zero should be 3. The reset location is written to a second time and the status register is read. The reset function is considered operational if the value of bits one and zero of the status register is restored to one. If the test fails, either the reset function is not operational or the status register has failed.

Test Sequence Number: 2. All further testing depends on the ability to correctly reset the NE.

Sub-block: Dual Port Ram

Parts List: U4138

Test Description: A simple read/write pattern test is performed on the 2 Kbytes of the dual port RAM. To pass this test, the pattern read from a given location must be equal to the pattern that was written to it. Since byte 0 of the DP RAM performs a special control function for the NE, it is exempt from this test. A failure of any part of this test means replacing the device.

Test Sequence Number: 3. This test only depends on Test 1.

Sub-block: Dual Port Ram Contention Arbitration Capability

Parts List: U4073

Test Description: Since the on chip contention circuitry is faulty, additional logic in the form of this device was added to perform arbitration when both the PE and the NE try to write to the same location on this device at the same time. To test that the contention logic is working, the same location on both sides of the DP RAM must be accessed simultaneously. First, the local timer is read. The *verify_contention* macro is executed. This causes the global controller to lock out DPCOMM0 for approximately 64 μ seconds. Therefore, if the VME side of the DPCOMM0 is accessed now, 64 μ seconds should elapse before it receives a Dtask. After reading this location (the value read is irrelevant), the local timer is again read. If 64 μ seconds have elapsed, the contention logic is operating correctly. The time which actually expires is reported as part of the test results.

Test Sequence Number: 5. This test depends on the DP RAM test and the test for the Global Controller.

Sub-block: Receive FIFO

Parts List: U0121, U0125, U1421, U1425, U1205, U1417

Test Description: One byte is written to each of four DP RAM locations (DPDATA0-DPDATA3). The *deliver_dpram* macro is executed which causes these bytes to be written in order into the Receive FIFO. The receive FIFO is read as a long word and the four bytes compared to those written to DP RAM. If the corresponding bytes are equal, the Receive FIFO is operational. If it fails, the Receive FIFO is not operational or the DP RAM interface on the Voted Data Bus which it shares with the Receive FIFO is faulty. There is no software test to differentiate between these two possibilities. Replace the Receive FIFO and retry.

Test Sequence Number: 6. This test must follow the test for the Global Controller and the test of DP RAM.

Sub-block: Transmit FIFO

Parts List: U0121, U0125, U1421, U1425, U1409, U1413

Test Description: The macro *xmit_to_dpram* sends data from Data Width Converter on the VME bus through the Transmit FIFO to DP RAM through the Debug Wrap Buffer. This wrap test is performed by writing a known long word pattern to the Transmit FIFO and then executing the macro *xmit_to_dpram* four times. The contents of DP RAM 0 is compared to the corresponding byte in the long word pattern written to the Data Width Converter. If all four patterns match, the Data Width Converter, the Transmit FIFO and the Debug Wrap Buffer are considered operational. In this case, the same test is performed using the *wrap_vme* macro. This macro causes the data from the Transmit FIFO to be transferred to the Receive FIFO, thereby exercising some additional data paths in this interface. If this exercise produces no errors the result is a passing score for the Transmit FIFO. If some of the patterns in the *xmit_to_dpram* test match but others do not, the corresponding register(s) in the Data Width converter are failed. If none of the patterns match, either the Transmit FIFO has failed or the Debug Wrap Buffer has failed. In this case a test to verify the operation of the Class FIFO is performed. This test is described below. However, it uses the Receive FIFO instead of the DP RAM as a repository of the wrapped byte. Thus if this test also fails, the Debug Wrap Buffer appears to be the failed device. If a new device results in the same test results, the Transmit and Class FIFOs are both faulty and must be replaced.

Test Sequence Number: 7. This test must follow the test of the Global Controller, DP RAM and Receive FIFO.

Sub-block: Class FIFO

Parts List: U1201

Test Description: This test uses the *wrap_serp_vme* macro to transfer a byte from the Class FIFO to the Receive FIFO. If the byte written equals the byte read, the Class FIFO is considered operational. If they are not equal, the device is faulty.

Test Sequence Number: 8. This test must follow the presence test for the global controller and the Receive FIFO.

Sub-block: Status Register

Parts List: U0117

Test Description: Following the execution of an NE-reset command, the value of the lower two bits of the status register should be 01 corresponding to an asserted value of CTS and a de-asserted value of DR. An asserted value of CTS corresponds to an empty Transmit FIFO. A de-asserted value of DR corresponds

to and an empty Receive FIFO. This part of the test is run implicitly in the reset function test. However, the ability to clear the CTS bit (bit zero) in the status byte read from this register can be tested also. This is accomplished by writing a series of long words to the Transmit FIFO and reading the Status register. CTS should be de-asserted when the Transmit FIFO is more than half full or after 129 long words have been written. It should remain in that state while the Transmit FIFO is filled to hold finally 256 long words. Next the macro *wrap_vme* should be executed enough times to transfer half the data from the Transmit FIFO to the Receive FIFO. This will require adjusting the message size with the *write_to_ftc* macro. Since the largest message which can be sent is 15 long words and the smallest message is 4 long words, this can be accomplished with two 15 word messages and one 12 word message. At this point, the CTS should be reasserted. After the contents of the full Transmit FIFO are transferred to the Receive FIFO, the contents of the Receive FIFO are read and compared to the outgoing data. Each of the 256 long words read from the Receive FIFO should match the corresponding word written out to the Transmit FIFO. When the Receive FIFO is empty, DR should be de-asserted.

Test Sequence Number: 9.

Sub-block: Debug Wrap Buffer

Parts List: U2701

Test Description: When tests for both the Class FIFO and the Transmit FIFO fail, this buffer is implicated. No unique test for this device is possible.

Test Sequence Number: N.A.

7.4.3. Functional Block: Network Element Data Paths

Sub-block: Data Paths through My FIFO and Opposite FIFO

Parts List: U2705 (Tri-Statable Pipeline Register), U1667, U1661, U1666, and U1669 (Debug Router), U0167 and U0149 (My FIFO), U0169 and U1163 (Opposite FIFO), U0244 and U1244 (Voter), U2163 (Synchronous Data Path Controller), U2244 (Asynchronous Data Path Controller), U2149 (Vote Mask Register)

Test Description: The purpose of this test is to determine whether or not the data paths through the FIFOs designated as My FIFO and Opposite FIFO are functioning correctly. Since the devices comprising the FIFOs cannot be fully isolated, the hardware comprising the entire data path is also tested both implicitly and explicitly by the test suite described here. Two data paths are exercised. The aggregate error information obtained during this process is then analyzed to identify as closely as possible the source of any errors. In the first test sequence, data is sent from the Transmit FIFO over My External Bus through the Debug Router to My FIFO, through the Voter and finally read back from the Receive FIFO. For this data path, the Vote Mask is set to exclude data from all channels except My FIFO in the voted result which is returned to the Receive FIFO. In the second test sequence, the data path is the same except the data path FIFO used is the Opposite FIFO. The Vote Mask is set to vote only data from the Opposite FIFO. (The voter PALs only allow simplex "voting" of data from these two particular FIFOs). For both tests the messages sent are the same: four words which contain bit patterns for byte wide "marching ones" through a field of zeroes and "marching zeroes" through a field of ones. Each test answers the following two questions: (1) Does the data read match, bit for bit, the data written? (2) Were any voter syndromes registered against the

data path FIFO in use? If all patterns read match the patterns written and no voter syndrome errors are reported against the channel under test, the devices in these data paths are functional. If no data is delivered to the Receive FIFO for either path, the functional blocks suspected of being faulty are the Synchronous or Asynchronous Controllers. If pattern errors are detected on both paths but no voter syndrome errors are detected, then devices in the Tri-Statable Pipeline Register or the Debug Router may be faulty. If a pattern mismatch occurs on only one data path, then the FIFO in that path or its associated pipeline register may be faulty. If syndrome errors are reported on both data paths, then the Voter or the Vote Mask Register may be faulty. If any errors are detected, the raw test results of this test are displayed on the attached monitor in tabular form.

Test Sequence Number: 10.

Sub-block: Data Paths through Left FIFO and Right FIFO

Parts List: U1667, U1661, U1666, and U1669 (Debug Router), U0161 and U0153 (Left FIFO), U0165 and U1149 (Right FIFO), U0244 and U1244 (Voter), U2163 (Synchronous Data Path Controller), U2244 (Asynchronous Data Path Controller), U2149 (Vote Mask Register)

Test Description: This test and the data analysis are performed in exactly the same manner as the test for the Data Paths through My FIFO and Opposite FIFO except that a different Vote mask is used. This test is only performed when no errors are detected on the previous data paths test suite. The Vote Mask used to test the Left FIFO includes My FIFO, Opposite FIFO and Left FIFO. For testing the Right FIFO, the mask is changed to include Right FIFO and exclude Left FIFO. If any errors are detected by this test, the raw test results of both sets of data path tests are displayed on the attached monitor in tabular form.

Test Sequence Number: 11

Sub-block: Voter Error Detection Capability

Parts List: U0244 and U1244 (Voter), U2149 (Vote Mask Register), U3877 (Syndrome Accumulator)

Test Description: The purpose of this test is to determine if the error detection capability of the Voter is functioning properly. This test is only performed if the Data Paths test suites have detected no errors in at least three of the the data paths. Depending on the number of working FIFOs, all possible configurations are tested. If all four FIFOs are fully functional, then the quadruplex and three triplex configurations are tested. If only three FIFOs are operational, then only one triplex configuration is tested. Each configuration is tested separately. One at a time, each channel of a given configuration is designated as the channel under test. The channel so designated is sent a corrupted message while the other channels receive congruent copies of the valid message. The messages are selected so that error detection is tested for every bit in eight bit wide voted data path. Furthermore, both possible error types are tested in each bit position, i.e. a correct value of zero and an erroneous value of one and vice versa. Error insertion is accomplished by first writing a valid message to all the FIFOs, then clearing the FIFO under test and sending all FIFOs the corrupted message. The FIFO under test now holds only one message, the corrupted one while the other FIFOs have a valid message followed by the corrupted message. Since the first message in the FIFOs are voted together, this procedure correctly inserts an error in the FIFO under test. Having thus inserted an error, the message is voted and delivered and the resulting voter syndrome information is examined. If, in all cases, an error is recorded against the channel under test, the error detection hardware is deemed to be functioning correctly. Furthermore, if, in all cases, the voted value of the message correctly

masks the error, the voter hardware is deemed to be functioning correctly. Otherwise, a malfunctioning component exists among the devices listed above. In this case, the results of the test are displayed in tabular form on the attached monitor.

Test Sequence Number: 12

Sub-block: Message Reflection Multiplexer

Parts List: U2657, U2661, U2665, U2669

Test Description: The purpose of this test is to verify that the special data paths involved in Class 2 exchanges are operating correctly. In particular, this test exercises the reflection path through the multiplexer which performs the second round of the Class 2 exchange. Each reflect path is tested in turn. A message is written to the data path FIFOs with the *wrap_to_dp* macro. The FIFOs which are not under test are then cleared. This is followed by the *reflect_from_X* macro, where X is either A,B,C, or D, depending on the reflect path under test. The message is then voted and read from the Receive FIFO. If the patterns match and no voter syndrome errors are reported, the reflect function is working properly. Any pattern mismatches or syndromes errors are indications of faulty hardware along the reflection data path and this information is therefore displayed in a message on the attached monitor.

NOTE: The microcode does not correctly execute the *reflect_from_X* macro, so this test cannot be performed.

Test Sequence Number: 13

7.4.4. Functional Block: Network Element Global Controller

Sub-block: The Global Controller

Parts List: U1473, U2773, U1491, U1485, U0181, U0185, U1477, U0177, U0173, U2725, U3869, U2777, U2781, U3885

Test Description: Despite its complexity, there is very little visibility into the operation of this functional block from the PE. A presence test can be performed and the results read back from DP RAM by executing the macro *write_pattern*. The Global Controller is considered "active" if the correct pattern is written to bytes 0 and 1 of the DP RAM. Another macro, *verify_counter*, causes the Global Controller to load its counter with 255 and to count down to zero. However, the successful execution of the presence test and the countdown test does not mean that the Global Controller is fully operational. Massive failures of other tests implicate the Global Controller. However, without detailed knowledge of its operation, ascertaining which devices are faulty is not possible.

Test Sequence Number: 4

Sub-block: ISYNC Test

Parts List: All NE components except the Scoreboard

Test Description: The purpose of this test is to verify the operation of the Global Controller in performing the initial network element synchronization, ISYNC. Even though the synchronization taking place is trivial, because synchronization with oneself is true by definition, the Global Controller does not "know" this fact. Therefore, it performs the synchronization as if it were trying to synchronize the quadriplex NE configuration, utilizing all the same logic and state changes required in a "real" synchronization. At the end of ISYNC, the NE reports the channels

with which it is synchronized in a dual port RAM location. In this case, it should be synchronized with itself *and* the three other NE channels, since the debug router is programmed to transfer data from the simplex channel conducting the test to all data path FIFOs, not from the external optical network. Following ISYNC, the NE is in synchronous mode and therefore will no longer respond to commands placed in the dual port RAM. Instead, it responds only to message sending commands generated by writing to the Class FIFO with the desired message size and class. The presence of messages delivered to the Receive FIFO are detected by reading the Status Register. To return to debug mode the NE must be reset.
Test Sequence Number: 14

7.4.5. Functional Block: The Scoreboard

Sub-block: Message Size Test

Parts List: U3234, U3244

Test Description: The purpose of this test is to verify to operation of the Scoreboard in sending packets of every allowable length. The Scoreboards of the various NEs communicate with each other by means of System Exchange Request Packets (SERPs). The SERPs contain information on the contents of the Class FIFO on each NE as written by its associated PE. In this simplex mode of pseudo-synchronous operation, the SERP packets are all identical and therefore the voted SERPs processed by the Scoreboard should always cause the message sent by the PE to be delivered. One at a time, messages of length 16N bytes (where N has a value from 1 to 15 inclusive) are written to the Transmit FIFO. The size and class of the message are then written to the Class FIFO. The Status Register is polled until it indicates the presence of a message in the Receive FIFO. The delivered message is then read into a buffer. The contents and size of the message received is compared with the message that was sent. If there is any disagreement, this result is reported in an error message displayed on the attached monitor. The voter masks are set so that this test is performed for every possible configuration of channels over the minimum fault masking number which in this case is three, provided that at least this many channels are deemed to be working correctly. Thus even if only three channels are working correctly, the quadruplex and four triplex configurations are still tested, since this is a normal operational condition. This test is integrated with the following test for Message Size to exercise every possible combination of message class and message size with every voting mask that provides a fault masking group.

Test Sequence Number: 15, the NE must be in synchronous mode following ISYNC

Sub-block: Message Class Test

Parts List: U3234, U3244

Test Description: The purpose of this test is to verify to operation of the Scoreboard in sending packets of every allowable class. It is similar in execution to the Message Size Test. In fact, these two tests are merged to allow every possible combination of class and size to be written to the Class FIFO. The contents and size of the message received is compared with the message that was sent. If there is any disagreement, this result is reported in an error message displayed on the attached monitor. In implementing this test and the previous test for Message Class, every possible combination of message class and message size is exercised with every voting mask that provides a fault masking group.

Test Sequence Number: 16, the NE must be in synchronous mode following ISYNC

7.4.6. Functional Block: The Inter-Fault Set Communication Links

Sub-block: Optical Data Links and TAXIs

Parts List: U4209, U4229, U4245, U4266 (ODLs), U4914, U4214, U4363, U4250 (TAXIs)

Test Description: The purpose of this test is to verify the correct operation of the devices used in the C2 optical communication network. This test requires that the cables which usually connect the Optical Transmitters to other fault sets be instead connected to the Optical Receivers of the simplex channel itself. In this wrap-around mode, the debug router must be programmed to route data received optically to the data path FIFOs. The tests which exercise the data path FIFOs, ISYNC and the message class and size are repeated for this configuration.

Test Sequence Number: 17

7.4.7. Conclusions

While the series of NE self tests exercise much of its functionality, the testing procedures demonstrate some deficiencies in hardware architecture of C2's NE. The NE is a highly integrated hardware unit. In a few cases a specific IC can be identified as the source of an error. However, in most cases, a set of ICs are identified as the most likely source. Even worse, in a few cases, the actual source of the error is completely indeterminate.

These complaints are most obvious regarding the functionality of the Global Controller, which is pervasive. Since all the self-testing depends on the full functionality of the Global Controller, and since there is no way to fully verify the Global Controller itself, any failed test could potentially be due to faults in the Global Controller, comprising 14 devices, or due to bad connections between various other devices and the outputs of the Global Controller.

In the C2 NE, there is no microcode in the Global Controller to support fully independent standalone testing of the Scoreboard. The operation of the Scoreboard can only be observed during the synchronous operation of the system. However, it is possible to operate the NE in a pseudo-synchronous mode in which the NE is synchronized only with itself. In this mode, the correct operation of the Scoreboard can be inferred from various positive test results.

Complete end-to-end testing of the inter-fault set communication links can only be performed in conjunction with other NEs. However, it is possible to wrap the output of

the optical transmitter to the input of the optical receiver on the same NE. However, this requires operator manipulation of these cables. In this configuration, the operation of the TAXI chips and the Optical Data Links can be tested.

Because the C2 NE was not designed for testability many functions of the NE were not sufficiently modularized or accessible to test facilities. In many cases, it was difficult to devise tests of functional components because of the inaccessibility of the necessary information. In those instances, it was necessary to perform numerous tests and to analyze the entire set of test results rather than to straight-forwardly exercise a single function with a predictable result.

Finally, there is no functional level test of only the fault tolerant clock (FTC). Although the fault tolerant clock is exercised during the scoreboard testing, the level of testing is minimal. A fault in the FTC may or may not adversely effect the results of these tests depending upon the nature of the fault.

7.5. AFTA Maintenance

The AFTA is being designed under the assumption that two domains of maintenance activity will be used during its operational life. These are field maintenance and depot maintenance.

7.6. AFTA Line Maintenance Procedure

An overview of the maintenance time line is depicted in Table 7-1. The times shown to perform the various maintenance steps are extremely preliminary estimates. The AFTA architectural features relevant to the maintenance discussion are shown in Figure 7-4.

Operation	Estimated time reqd	Comments
Perform M-BIT	TBD	Initiated via CDU Identifies LRU, LRM, Bay on CDU
Open Bay	30 minutes	
Connect PIMA	2 minutes	
Download/Print Fault Log	2 minutes	
Replace LRM/LRU	30 minutes for LRM, 60 minutes for LRU	
Perform M-BIT	TBD	Retest entire AFTA via PIMA/CDU
Close Bay	30 minutes	

Table 7-1. AFTA Maintenance Time Line

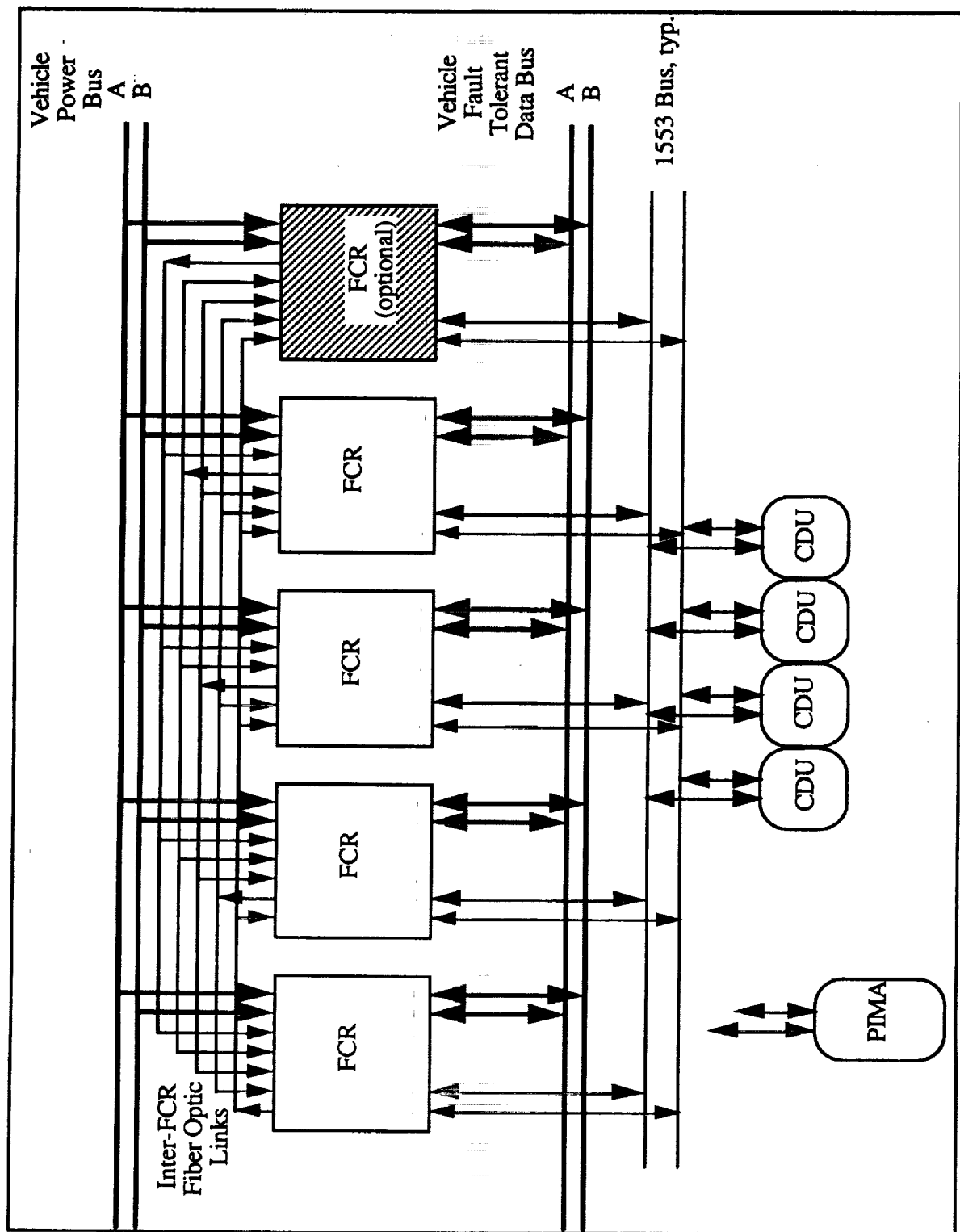


Figure 7-4. Maintenance-Related AFTA Features

M-BIT is initiated by crew chief or pilot via CDU or Portable Intelligence Maintenance Aid (PIMA) and is interruptible at any time via AFTA reset or power cycle. Reset of power cycle throws AFTA into I-BIT power-up sequence. M-BIT and I-BIT are disabled in critical vehicle modes such as taxi, flight, etc.

M-BIT indicates fault status of AFTA on CDU. If faults are found, CDU indicates LRU, LRM, and bay in which LRU can be found. The bay containing the faulted AFTA LRM/LRU is opened by the maintenance crew. Vehicle-specific tools will be required to achieve this. Once the vehicle bay is open, the maintenance crew member can connect a PIMA to a port on the outside of the AFTA LRU (Figure 7-5). PIMA ports will also exist elsewhere in the vehicle which will not require opening of vehicle bays. According to LH doctrine, each vehicle possesses a PIMA, whose primary purpose is to assist maintenance personnel in isolating vehicle faults and diagnosing problems; it is assumed that it will also be used for AFTA diagnosis and maintenance. The PIMA, essentially a largish ruggedized laptop computer, will be connected to the vehicle after each flight to interrogate system status. For the AFTA, the PIMA is capable of displaying and printing (via an AVIS rental car-like printer) the nonvolatile fault log maintained by the AFTA, sending the AFTA through I-BIT and M-BIT, resetting the AFTA, and resetting the nonvolatile fault log.

The AFTA system services are also responsible for logging LRM/LRU utilization information such as number of power cycles, elapsed power-on time, and other information determined to be of interest to maintenance personnel. This information will be maintained for each replaceable component, downloaded to the PIMA upon request, and will accompany defective modules back to the depot via the PIMA printout. In addition to its diagnosis function the PIMA will keep records, and store and display all flightline maintenance and logistics publications [MIL-HDBK-59].

To further facilitate maintenance, an annunciator panel located on the exterior of the LRU indicates which LRM inside is faulty. The panel contains one indicator for each LRM slot.

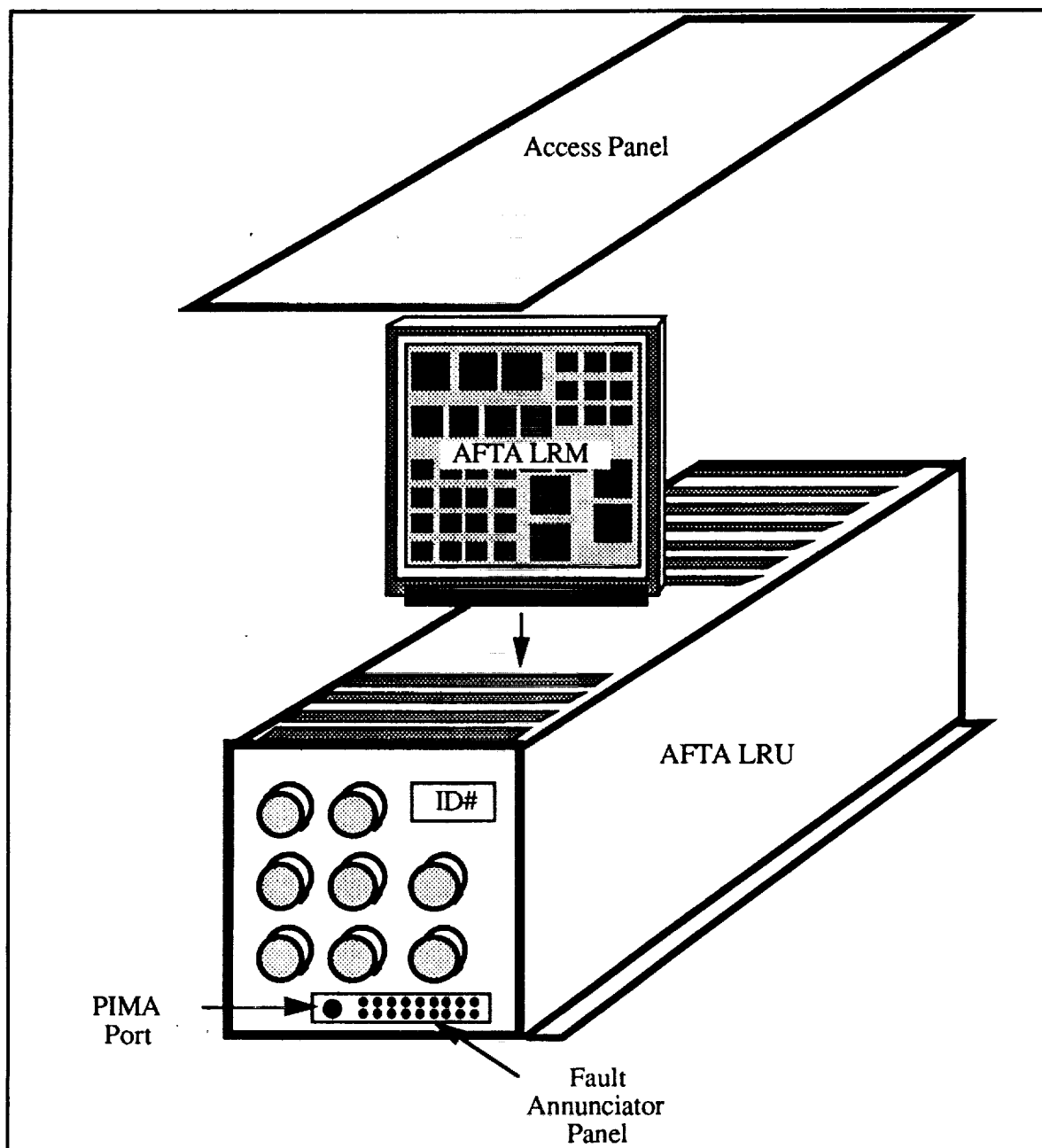


Figure 7-5. AFTA LRU

If the fault cannot be isolated to an LRM, the entire LRU must be replaced. This will probably require tools to remove the chassis and wiring. An example of such a fault would be failure of the intra-FCR backplane connecting the LRMs.

If the fault is isolated to one or more LRMs, as indicated by both the CDU, PIMA, and annunciator panel indicators, the maintenance technician opens the AFTA LRU by re-

moving the access panel. The access panel is sealed to the LRU chassis with manually operable closures to allow this to be performed without the use of special tools. The faulty LRM(s) is (are) removed using the injector/extractors attached to the LRMs; Allen wrenches will be required to unseat and seat the "wedge-locks" on the LRM cold edges from the chassis cold frame. Replacement LRMs are installed and secured using the LRM injectors and wedge-locks.

After replacement of the LRU or LRM(s) and before replacement of the access panel, the M-BIT is repeated, either from the CDU or the PIMA. Assuming that the M-BIT is passed, the LRU access panel is replaced and the bay is sealed.

It is possible that I-BIT, M-BIT, and C-BIT could be used to diagnose faults down to the integrated circuit level, and that this information could be recorded and printed out by the PIMA and accompany the defective LRM/LRU back to the depot. However, because it is necessary to confirm at the depot that the defective circuit was repaired before shipping it back to the field, some ATE will be required for depot testing, assuming the AFTA components cannot test themselves to the IC level at the depot.

8. Common Mode Fault Study

Most current fault tolerant architectures are primarily designed to tolerate random hardware faults. It is assumed that the probabilities that redundant copies of a computation suffer a fault at the same time are independent and uncorrelated. This is an accurate assumption for random hardware faults but a poor one for common mode faults such as those caused by software faults, because all copies of the execution will suffer the fault at the same time if the copies are identical. Software faults are but a specific class of common mode faults. Other sources of common mode faults are generic hardware bugs or design flaws, massive electrical upsets which overwhelm the fault tolerant power/clocking fault containment mechanisms, etc. More rigorously, a common mode fault is defined to be a fault that affects multiple fault containment region simultaneously or nearly simultaneously. Nearly simultaneous in the AFTA context means that the system has not recovered from a fault before the next fault arrives.

Under the AFTA program, a methodology for detecting and recovering from common mode faults in AFTA will be developed. In addition, a plan for verifying the effectiveness of these techniques will be formulated. In the event that application-specific information for the study is needed, the helicopter TF/TA/NOE application will be used as a context for the common-mode fault tolerance study.

8.1. Objective

The objective of this study is to develop a comprehensive methodology for reducing the probability of failure of synchronous redundant Byzantine resilient computing systems due to common mode faults. The methodology is to include techniques to avoid, remove, and tolerate common mode hardware and software faults, the identification of means to verify the effectiveness of the common mode fault avoidance/removal/tolerance (CMFA/R/T) techniques, and a timetable for inserting or developing appropriate CMFA/R/T technology into AFTA.

8.2. Approach

The study comprises four phases. Phases 1 and 2 comprise technology surveys, phase 3 comprises evaluation and planning, and phase 4 comprises initiation of the plan.

8.3. Enumeration of Common Mode Fault Sources

Common mode faults and their sources are extremely diverse. They can be classified in the same way that all faults are classified in the document "Dependability: Basic Concepts and Terminology" [Lap90]. They can be classified according to three main viewpoints which are their nature, their origin and their persistence. The three viewpoints are not mutually exclusive.

8.3.1 Classification by Nature

Faults may be accidental in nature, i.e., they appear or are created fortuitously. Or they may be intentional in nature, i.e., they are created deliberately.

For AFTA, intentional faults, e.g. Trojan horses, time bombs, viruses, will not be considered since they are related to secure systems. Security is currently not a requirement for AFTA applications although it may be at some future point in time.

8.3.2 Classification by Origin

This is further divided into three viewpoints which are not necessarily mutually exclusive:

1. Phenomenological Causes

- physical faults, which are due to adverse physical phenomena;
- human-mode faults, which result from human imperfections.

2. System Boundaries

- internal faults, which are those parts of the system's state which, when invoked by the computation activity, will produce an error;
- external faults, which result from system interference caused by its physical environment, or from system interaction with its human environment.

3. Phase of Creation

- design faults, which result from imperfections that arise during: the development of the system (from requirement specification to implementation), subsequent modifications, or the establishment of procedures for operating or maintaining the system;

- operational faults, which appear during the system's exploitation.

8.3.3 Classification by persistence

1. Permanent Faults

- their presence is not related to internal conditions such as computation activity or external conditions such as the environment.

2. Temporary Faults

- their presence is related such conditions and as such they are present for a limited amount of time.

Since intentional faults are excluded from the current scope of work, there are only 16 possible sources of faults that must be considered. These are all the possible combinations of the remaining four viewpoints. Of these the physical, internal, operational faults can be tolerated by using hardware redundancy. All other faults can affect multiple fault containment regions simultaneously. These are the sources of common mode faults. However, only some of these fault classes are meaningful. These are tabulated in Table 8-1. Of these, the interaction faults which arise from the interaction of the computer system with its human environment, e.g. an operator, will not be considered here since the man-machine interface is outside the scope of the AFTA's use as an embedded control system.

Phenomenological cause		System Boundary		Phase of Creation		Persistence		Common Mode Fault Label
Physical	Human made	Internal	External	Design	Operational	Permanent	Temporary	
X			X		X		X	Transient (External) CMF
X			X		X	X		Permanent (External) CMF
	X	X		X			X	Intermittent (Design) CMF
	X	X		X		X		(Permanent) Design CMF
	X		X		X		X	Interaction CMF

Table 8-1. Classification of Common Mode Faults

Using this methodology, then, only 4 sources of common mode faults need to be considered for AFTA:

1. Transient (External) Faults which are the result of interference to the system from its physical environment such as lightning, High Energy Radio Frequencies (HERF), heat, etc.
2. Permanent (External) Faults which are the result of system interference caused by its operational environment such as heat, sand, salt water, dust, etc.
3. Intermittent (design) Faults which are introduced due to imperfections in the requirements specifications, detailed design, implementation of design and other phases leading up to the operation of the system. These faults manifest themselves only part of the time.
4. (Permanent) Design Faults are introduced during the same phases as intermittent faults, but manifest themselves permanently.

If the relative likelihoods of these four classes of common mode faults were known, one could apportion the efforts in dealing with them appropriately. However, the models to predict the occurrence of design faults do not exist or are not mature enough to be of any practical value to AFTA. Similarly, the rates of occurrence of transient faults and permanent external faults are very much dependent upon the operational environment. Therefore, the relative rates of occurrence of the four classes of AFTA common mode faults cannot be predicted with any certainty. Experience suggests that all of these are sufficiently likely to be of concern to the designers of AFTA.

8.4. Enumeration of Common Mode Fault Avoidance, Removal, Tolerance Techniques

There is a wide range of techniques available today to prevent introduction of CMFs into a Byzantine Resilient fault tolerant computer, to remove CMFs before such a computer is put into operational use, and to detect and recover from CMFs that do occur during operational use. As such, the techniques and tools can be classified into three major categories: Fault Avoidance, Fault Removal and Fault Tolerance. These classifications are described in this section while their effectiveness and suitability for inclusion in AFTA will be discussed later.

8.4.1 Common Mode Fault Avoidance

These techniques and tools are used from the requirements specifications to the design and implementation phases and result in fewer CMFs being introduced into the computer system. An unprioritized list of these techniques and tools, without regard to their effectiveness in preventing CMFs or applicability to AFTA, is as follows.

8.4.1.1. Formal Methods

These are mathematically based techniques for specifying, developing, and verifying computer systems with strong emphasis on consistency, completeness and correctness of system properties. Formal methods have been applied at various levels of specification and design and to a diverse set of hardware, software and algorithmic parts of fault tolerant computers. Some of the example applications include the following.

Microprocessor Design: Viper [Cohn88], FM8501 [Hun86], Mini Cayuga [Sri90];

Algorithm Specification and Implementation: Interactive Consistency and Oral Messages 1 [Bev90], [Bic90];

Fault Tolerant Clock Synchronization: [Dal73], [Lam85], [But88], [Rus89];

Specialized Hardware: Communicator-Interstage [Klj88];

Software: Real Time Kernel [Spi90], Ada Formal Verification [Gua90], Formal Specification [Goe91]

Reliable Computing Platform: [DiV91].

8.4.1.2. Formally Verified Components

Use of hardware and software modules that have previously been verified or have been developed using formal methods can reduce the incidence of CMFs in these parts of a fault tolerant computer. Examples of such components include microprocessors such as VIPER, VIPER2, FM8502, Mini Cayuga, Floating Point Units, and Real Time Kernels.

8.4.1.3. Mature Components

Use of hardware and software modules that have been widely used over a long time period and whose performance has been monitored and analyzed for correctness of opera-

tion can also cut down the incidence of design faults. Examples of such hardware modules include popular microprocessors such as Motorola 68020, Intel 80386, etc., floating point coprocessors, memory management units, and Ethernet and VMEbus controllers. Examples of mature software modules include Ada Run Time System and CAMP (Common Ada Missile Packages) software libraries [CAMP]. CAMP products have been developed under contract to the US Air Force Armament Test Laboratory, Eglin Air Force Base, Florida and are available from Data & Analysis Center for Software [DACS]. CAMP products consist of Parts, Armonics Benchmarks, and Parts Engineering System (PES). The CAMP Parts are 444 reusable Ada components organized into 35 Top-Level Computer Software Components (TLCSCs) which contain 137,000 source lines of Ada code (including comments, package specifications, package bodies, and test code). The CAMP Armonics Benchmarks are used to evaluate Ada and processor implementations in the armonics domain. The benchmarks represent typical armonics applications and include missile operational parts as well as support parts from the mathematical domain. The tests establish the "correctness" of compiler implementations and measure performance in size and speed of generated code. The CAMP PES is a catalog that provides a means of identifying and retrieving reusable software parts.

8.4.1.4. Design Automation Tools

These are tools and techniques that can help automate parts of the hardware and software design cycle. By replacing a labor intensive design process with automated tools, the incidence of human errors can be reduced. In the software arena, more than 50 different CASE (Computer Aided Software Engineering) tools are available that provide different levels of automated software generation. The Draper CASE tool has been used, among other applications, to produce Boeing 737 autoland code in Ada starting from a high level control law specification. The Ada code was compiled and integrated with the existing system software on the AIPS Fault Tolerant Processor without any modifications.

In the hardware arena, VHDL (VHSIC Hardware Description Language) is becoming widely available to describe hardware designs at various levels of abstraction, from a high level functional description to all the way down to the gate level.

A suite of tools, generally known as Silicon compilers, can be used to convert VHDL or other high level design descriptions through various levels of detailed hardware design, right down to the Silicon implementation with some help from the human designer. One such suite of tools is the Silicon 1076 compiler from LSI Logic, Inc. which interfaces with

a VHDL design description at the top and produces a silicon chip at the end of the design cycle.

8.4.1.5. Architectural Considerations

Human errors are more likely when dealing with complex systems and unconventional concepts. In a fault tolerant computer, concepts that add to the design complexity of a conventional Von Neumann uniprocessor computer architecture are redundancy management and distributed and parallel processing. Examples of the additional complexities that a designer faces are: fault containment, error containment, synchronization of redundant processes, communication between redundant processes, synchronization of and communication between distributed/parallel processes, all of these in the presence of one or more faults, detection, isolation and recovery from faults, and so on.

If the design complexity can be reduced then the incidence of human errors can be reduced. Some of the fault tolerance concepts can be stated simply and precisely using a mathematical formalism. These include the requirements for synchronization, agreement and validity. Other concepts that can be stated precisely include requirements for fault containment and error containment. Because of their simplicity fault tolerant computers that are based on these concepts and implement these requirements are likely to contain fewer design errors. (It should be noted here that not all fault tolerant computers implement these requirements.) There is an added benefit in the design verification and fault removal phases of basing designs on precisely stated requirements.

Another architectural consideration is the hiding of the design complexity. For example, certain architectures implement fault tolerance in such a manner that the virtual architecture apparent to the applications programmer and the operating system programmer appears to be that of a conventional non-redundant computer. The complexities of a redundant architecture are made visible only to the tasks that must deal with detection and isolation of faults and recovery from faults.

Similarly, the complexities of distributed/parallel processing can be hidden from most of the software designers by providing layered communication protocols such as the ISO OSI models.

8.4.1.6. Design Diversity

Design diversity is the concept of implementing different layers of a redundant system using different designs starting from a common set of specifications. The concept can be applied to hardware, software, programming language, design development environment and other design activities. This approach can potentially eliminate many common mode design faults since each redundant layer uses a different design. Some design faults such as those that result from an incorrect interpretation of ambiguous specifications could still find their way in multiple or all designs. Design diversity is listed here as a fault avoidance rather than a fault tolerance technique since it purports to confine each design fault to a single fault containment region, thereby avoiding a common mode fault.

When redundant hardware and/or software elements are implemented using different designs, bit-wise exact consensus cannot be guaranteed between the outputs of redundant processors. However, it is still possible to provide a Byzantine resilient core fault tolerant computer in which design diversity is used for applications programs.

8.4.1.7. Use of Standards

Over the years, a number of standards have been developed for the design of computer systems. Although the primary motivation for the development of standards is ease of interoperability, logistics, maintainability, reduced cost, and so on, one of the side benefits of using standards is the reduction of design errors. Standards usually result in detailed, precise, and stable specifications that can be adhered to in the design phase and verified against in the verification phase. The design errors that are normally introduced due to ambiguous or changing specifications can potentially be eliminated by the use of standards.

Examples of standards include bus protocols such as MIL-STD 1553, PI bus, High Speed Data Bus and processor Instruction Set Architectures such as MIL-STD 1750 and more recently the JIAWG ISAs such as the Intel 80960 and the MIPS R3000. Software standards include the MIL-STD-1815, more commonly known as the Ada language. The advantages of mature, precise and detailed specifications are not limited to military standards alone. Commercial products, though not standards, per se, can become de facto standards. VMEbus is such an example of a backplane bus standard.

8.4.1.8. Good Software Engineering Practices

Common mode faults caused by software are probably the largest single source of failure in a redundant computer system. Many software errors can be avoided by following well established software engineering practices. These practices include adherence to the waterfall software development methodology, that is, an orderly development of requirements, specifications, detailed design, code, unit test, module test, integration, and system test, with traceability of requirements from beginning to end. Rigorous configuration control and documentation, such as that specified by MIL-STD 2167A, are also considered an integral part of this methodology. Other good software engineering practices include software quality assurance reviews, use of Higher Order Languages such as Ada, modular code, code reuse, and layering or hierarchical structuring of code such as the 7 layers of the Open Systems Interconnect model for intercomputer communications.

It should be noted here that many of these software engineering practices overlap with other fault avoidance techniques discussed in this section that can be used to avoid not just software but also other common mode faults. For example, software quality assurance reviews can be considered a part of design reviews which are applicable to hardware as well as software. Similarly, code reuse overlaps with use of mature components, and so on.

Since software development is a labor intensive process, one of the good software engineering practices deals with the training and qualification of people. It is important to assign software development duties to people who have been fully trained in the desired software language, tools, and development methodologies and possess an appropriate type and amount of experience in the relevant activities.

One aspect of personnel qualification that might be considered controversial is the "quality" of people. It has been our experience that not all software designers who are equally trained and experienced produce equally "good" software where quality of software is measured by the number of errors. The difference between the best and the worst designers can be an order of magnitude in the number of errors. Therefore, track record of software developers as much as training and experience must be given consideration if the goal is to produce high quality software.

8.4.1.9. Conservative Hardware Design Practices

Conservative hardware design practices can help keep a healthy margin of safety between the worst operational conditions and the actual limits of operation of the computer

system. The MIL-SPEC 883C operational temperature range, for example, is -55°C to $+125^{\circ}\text{C}$ even though such extremes are probably never reached in actual use. Military design guidelines also call for derating of electronic parts by a certain margin depending on the environment. For example, the current outputs of drivers, wattage rating of resistors or maximum voltages of capacitors may be reduced by 20 to 50 percent to keep the operational parameters well within the maximum design values. Use of military qualified parts, as specified by MIL-STD 38510 or SMDs (Standard Military Drawings), ensures that the part designs meet the functional and electrical specifications and that the parts have been manufactured and screened as specified by MIL-SPEC 883C.

Similar to the good software engineering practices discussed in the previous section, the hardware logic can also be designed in accordance with conservative design rules developed over the years. Some of the example rules are the use of synchronous designs and the avoidance of metastable states. Timing verification on synchronous designs is much easier than for asynchronous designs. The worst case performance of a synchronous design always occurs at the longest propagation delay; a properly designed synchronous circuit will still function using delta (approaching zero) delays. Synchronous circuits typically depend on inputs which are edge-sensitive. Edge-sensitive inputs should only be driven by a signal that is guaranteed to be free of unwanted glitches. Acceptable signals are clocks, outputs from registers, or flip-flops (not latches), and combinatorial circuits specifically designed to be free of glitches under all conditions.

The state of a synchronous design should be predictable at all times following power-up or reset. All state elements, except simple data registers, should be initialized by the reset process. Trap states in a state machine can be avoided by defining unconditional transitions from unused states into the set of defined states. While the machine should not normally enter an unused state, prevention of trap states ensures that if it does enter such a state, through a metastability problem or a single-event upset, the system will eventually recover.

Metastability is another cause of common mode hardware faults that can be easily avoided by adhering to . Metastability is caused by inputs changing within the setup and hold time of a flip-flop and can cause finite-state machines to enter unwanted or undefined states. Asynchronous inputs (those that might change within the setup and hold time) are easily synchronized by passing the signal through a synchronizing flip-flop stage.

8.4.1.10. Shielding, Packaging and Thermal Management

An appropriate amount and type of shielding and packaging can keep the HERF, Single Event Upsets (SEUs), and lightning from interfering with the correct operation of the computer system. Similarly, proper packaging and cooling can keep the dust, saltwater, sand and other foreign matter outside and also dissipate the heat generated internally to the outside. Proper packaging techniques can also assure that the hardware will survive the expected shock and vibration environment. Mil-E-5400 provides the military specifications for thermal management and shock and vibration design requirements.

8.4.2 Common Mode Fault Removal

Faults that slip past the design process can be found and removed at various stages prior to the computer system becoming operational. The fault removal techniques and tools include the following.

8.4.2.1. Design Reviews

Traditionally, informal design reviews and code walk-throughs between engineers and peers as well as formal design reviews such as PDR (Preliminary Design Review), CDR (Critical DR), SRR (Software Requirements Review) by supervisors and managers have been used to uncover gross design and implementation errors. The management reviews also check the compliance of the design with the intent of the high level requirements and specifications, which may not always be stated unambiguously and precisely.

8.4.2.2. Simulations

Simulations have been used at various levels to check compliance with design goals. Functional and timing simulations have been a must before ASICs (Application Specific Integrated Circuits) can be fabricated. VHDL can now be used to perform behavioral simulations before a function is even translated into an electronic circuit. VHDL can also be used to perform more detailed, lower level simulations as behavioral boxes are replaced by detailed circuit designs, all the way down to transistor characteristics.

8.4.2.3. Testing

For software, testing rather than simulation has been the traditional technique for uncovering design faults. Unit tests, module tests, functional tests, and code reading have

been used extensively in the past to verify correctness of software. Additionally, structural analysis tools can also be used to analyze static software behavior.

More recently, a novel approach to testing software, called back-to-back testing, has been advanced. It involves comparing outputs of the program under test against a functionally identical program that has been produced from the same specifications as the target program but by a different programming team and/or in a different language. This is similar to N-version programming in that multiple versions of a program are produced. However, after the testing phase is completed, only one of the versions is chosen for operational use. Any mismatches at the outputs can be traced to a design or coding error in one of the versions, an incorrect or different interpretation of specifications by different programming teams or a difference due to round-off errors.

Testing with oscilloscopes, logic analyzers and probes has been the traditional hardware debugging technique. With the advent of VLSI ASICs, very little visibility can be obtained inside these chips with these tools. Designing chips with testability such that all the internal nodes can be tested by applying test vectors (test inputs) and observing the outputs, has become a field in its own right. Scan-path is now a standard technique to make ASICs testable. Similarly, automatic generation of test vectors for ASICs is significantly more advanced than automatic generation of test inputs for software modules.

8.4.2.4. Fault Injection

Insertion of faults in an otherwise fault-free computer system that is designed to tolerate faults is a powerful technique to exercise redundancy management hardware and software that is specialized, error-prone, difficult to test and not likely to be exercised under normal conditions, i.e., likely to stay dormant until a real fault occurs. Fault insertion techniques can also be used to operate the system in various degraded modes which are expected to be encountered in operational life of the system. Degraded mode operation stresses not only fault handling and redundancy management aspects but also task scheduling, task and frame completion deadlines, workload assignment to processors, inter-task communication, flow control, and other performance-related system aspects. Fault insertion exposes the weaknesses in the hardware and software design, the interactions between hardware and software, and the interactions between redundancy management and system performance. It is an accelerated form of testing the hardware, software and the system, analogous to "bake and shake" testing of hardware devices.

Different types of tools have been developed to insert faults at various levels to stress the fault tolerance capabilities of computer systems. Draper has successfully used a pin-level hardware fault injector for the past 10 years to uncover subtle design errors in the FTMP, FTPs and AIPS. A similar tool has also been developed by Laboratoire d'Automatique et d'Analyses des Systemes (LAAS) of the French National Center for Scientific Research (CNRS) in Toulouse and used to evaluate a railway signal control computer.

Carnegie Mellon University has developed FIAT to insert memory faults. A memory mutation technique has also been used independently at Draper to stress the AIPS FTP redundancy management software. Chalmers University in Sweden has experimented with Californium as a radiation source to test a self-checking microprocessor pair.

Fault insertions at higher levels such as module, link, and fault containment region have also been used at Draper for the purposes of design verification.

8.4.2.5. Discrepancy Reports

A Discrepancy Report (DR) is filed any time anomalous or unexpected behavior of hardware, software or the system is encountered. A DR deals with the observed symptoms which may eventually be traced to one or more specific design, coding, manufacturing or other problems. A DR log can be started as soon as the first phase of testing is begun. This will normally be a unit test for software and module test for hardware. Alternatively, the log may be deferred until a unit/module has passed an acceptance test to the satisfaction of the designer. Delaying the log to this point can cut down the paper-work associated with the relatively large number of error symptoms which is a normal part of initial debugging. The risk with delaying the start of the log is that if the designer is not methodical in resolving all the observed discrepancies then causes of these errors may be left in the unit/module if the unit/module acceptance test does not produce the error.

In any case, the DRs are logged through all subsequent phases of testing, integration, and verification and validation activities. Once a DR is traced to an underlying cause or causes, and the problems are successfully resolved, the DR can be closed out after recording the cause(s) and the fixes made. Figure 8-1 shows a typical format for a Discrepancy Report. It should, at a minimum, describe originator, date, problem category, software identification (if known), hardware identification (if known), document identification, other related DRs, description of the symptoms or occurrence of an event, conditions, and con-

jecture on possible causes. Other fields in the DR form that need to be eventually filled out include analysis of cause and effect, recommended solution, disposition, and verification and close-out. Each DR must also be signed by responsible engineers and managers.

It should be pointed out here that a Discrepancy Report is more comprehensive than a "Bug Report" that is normally associated with the software testing phase. Typically, a Bug Report is filed when a bug, i.e., a cause of the software error is discovered. A DR precedes the discovery of the cause. It is filed when an anomalous behavior is discovered whether or not its cause can be immediately determined. Furthermore, it is not limited just to software but applies to hardware and the system as well. A further important distinction between a DR and a bug report is that a DR may eventually lead to the discovery of many related errors. Typically, in early phases of the system integration several hardware and software design errors, manufacturing defects, and subtle interactions conspire to produce bizarre system behavior. As each error or defect is found and corrected, symptoms change and become less or more bizarre due to the masking effect of one error on another. Sometimes the error symptoms disappear altogether due to a subtle change in timing of events. This is where DRs become quite useful in systematically accounting for abnormal behavior. For example, if at the delivery time, the system passes all the acceptance tests but not all the DRs have been successfully resolved, it implies that there could still be some latent errors in the system.

Discrepancy Reports bring a certain amount of discipline to resolving the observed problems. If the procedures for logging DRs are followed rigorously by all the engineers, programmers, and technicians working on the program, then the probability of removing all the known common mode faults is increased considerably. One no longer has to rely on the memory or methods of individual designer or tester to keep track of the known problems.

DRs can be used to collect statistical data necessary to predict the software reliability growth and other software reliability related metrics. The DR shown in Figure 8-1 can be expanded to record the data that will be necessary to plot the number of software errors discovered, the mean time between software error occurrence, relationship of errors to software units and lines of code, and so on. Under an internal R&D program, Draper is developing a system for automating the recording and searching of the DR database.


 DISCREPANCY REPORT The Charles Stark Draper Laboratory, Inc. Cambridge, Massachusetts 02139		DR NUMBER _____	SHEET _____ OF _____
PROJECT _____			
1. ORIGINATOR: _____	2. ORGANIZATION _____	3. DATE: _____	4. TELEPHONE #: _____
5. PROBLEM CATEGORY: <input type="checkbox"/> COMPUTER PROGRAM/DATA <input type="checkbox"/> HARDWARE <input type="checkbox"/> DOCUMENT <input type="checkbox"/> OTHER _____	6. SOFTWARE IDENTIFICATION (if known) CPCI: _____ CPC: _____ UNIT: _____ VERSION/REVISION: _____	7. HARDWARE IDENTIFICATION: (if known)	
8. DOCUMENT IDENTIFICATION: _____		9. RELATED DR: _____ <input type="checkbox"/> SUPERSEDED <input type="checkbox"/> MODIFIED	
10. DESCRIPTION (OCCURRENCE):			
11. CONDITIONS:			
12. POSSIBLE CAUSE:			
13. ANALYSIS OF CAUSE AND EFFECT:			
SIGNATURE _____ ORG _____ DATE _____ TEL. _____			
14. RECOMMENDED SOLUTION:			
SIGNATURE _____ ORG _____ DATE _____ TEL. _____			
15. DISPOSITION: <div style="display: flex; justify-content: space-between; margin-top: 10px;"> <div> <input type="checkbox"/> NO ACTION REQUIRED <input type="checkbox"/> ECR NO. _____ </div> <div> <input type="checkbox"/> ECP NO. _____ <input type="checkbox"/> OTHER _____ </div> </div> SIGNATURE _____ ORG _____ DATE _____ TEL. _____			
16. VERIFICATION AND CLOSE OUT:			
RESPONSIBLE ENGINEER _____		DATE _____	
TECHNICAL MANAGER _____		DATE _____	

Figure 8-1. Typical Discrepancy Report Format

8.4.2.6. Automated Theorem Provers

Automated theorem provers or mechanical checkers have been used to argue the compliance of an implementation with a set of specifications. In the course of showing the correspondence from one level to the next, one develops a set of arguments to convince the ATP that one formal statement follows from another. This usually leads to uncovering the errors in the correctness of implementation.

8.4.3 Common Mode Fault Tolerance

Common mode faults that are not removed prior to operational use of a computer system may eventually manifest themselves in the field. At this point the only recourse is to detect the occurrence of such a fault and take some corrective action. These are fault tolerance techniques and following is an unprioritized list of such methods.

8.4.3.1. Common Mode Fault Detection

Before a recovery procedure can be invoked to deal with common mode faults in real time, it is necessary to detect the occurrence of such an event. Many ad hoc techniques have been developed over the years to accomplish this objective. Most of these techniques can also be used prior to operational use of the system to eliminate faults. The difference is that in the fault removal phase, detection of a fault leads to some trap in the debugging environment while in the operational phase it will lead to a recovery routine. Similarly, fault removal techniques discussed in Section 8.4.2 can also be used to aid in the task of detecting faults in real time, albeit with a high penalty in performance.

a. Watchdog Timers

Watchdog timers can be used to catch both hardware and software wandering into undesirable states. They are typically used in the Processor Element but can also be employed in the Network Element. Neither hardware watchdog nor task timers unambiguously indicate the occurrence of a common mode fault. The syndrome in the failed channel of a physical fault is no different from that of a common mode fault. The syndromes across redundant channels must be compared in real time to determine the cause.

b. Hardware Exceptions

Hardware exceptions such as illegal address, illegal opcode, access violation, privilege violation, etc. are all indications of a malfunction. Again, syndromes across redundant channels must be correlated to distinguish between physical and common mode faults.

c. Ada Run Time Checks

Ada provides numerous run time checks such as type checks, range constraints, etc. that can detect malfunctions in real time. Additionally, user can define exceptions and exception handlers at various levels to trap abnormal or unexpected program/machine behavior.

d. Memory Management Unit

The Memory Management Unit can be programmed to limit access to memory and control registers by different tasks. Violations can be trapped by the MMU and trigger a recovery action.

e. Acceptance Tests

This is a very broad term and can be applied to applications tasks and various components of the operating system such as the task scheduler and dispatcher. The results of the target task are checked for acceptability using some criteria which may range from a single physical reasonableness check such as pitch command not exceeding a certain rate to an elaborate check of certain control blocks to ascertain whether the operating system scheduled all the tasks in a given frame.

It should be noted again that a physical fault can trigger any of these detection mechanisms just as well as a common mode fault. Therefore, it is necessary to corroborate the syndrome information across redundant channels to ascertain which recovery mechanism to use.

f. Presence Test

Presence test is normally used in FTPs and FTTPs to detect the loss of synchronization of a single channel due to a physical fault. However, it has also been modified to detect a total loss of synchronization between multiple channels of an AIPS FTP. This is an indication of a common mode fault. This technique can be extended to the FTTP as well.

8.4.3.2. Common Mode Fault Recovery

The recovery from CMF in real time requires that the state of the system be restored to a previously known correct point from which the computation activity can resume. This assumes that the occurrence of the common mode fault has been detected by one of the techniques discussed earlier and that its source has been identified.

- a. **Exception Handlers:** If a common mode fault causes an Ada exception or a hardware exception to be raised, then an appropriate exception handler that is written for that abnormal condition can effect recovery. The recovery may involve a local action such as flushing input buffers to clear-up an overflow condition or it may cascade into a more complex set of recovery actions such as restarting a task, a virtual group or the whole system.
- b. **Task Restart:** If the errors from CMF were limited to a single task and did not propagate to the operating system, then only the affected task needs to be restored and/or restarted with new inputs. The state can be rolled back using a checkpointed state from stable storage. Recovery is then effected by invoking an alternate version of the task using the old inputs assuming that the fault was caused by the task software. This is termed the backward recovery block approach. If the fault is caused by a simultaneous transient in all redundant hardware channels then the same task software can be re-executed using old inputs. This is termed temporal redundancy. Alternatively, forward recovery can be effected by restarting the task at some future point in time, usually the next iteration, using new inputs. This assumes that the fault was caused by an input sensitive software that will not repeat with new and different inputs.
- c. **Virtual Group Restart:** In case the CMF resulted in the loss of synchronization, then redundant channels must be re-synchronized before rollback can begin. Furthermore, the state of the virtual group must be restored before resuming computational activity.
- d. **System Restart:** Finally, if all else fails the whole system can be restarted in real time and a new system state established with current sensor inputs.

8.4.3.3. Performance Overheads of Common Mode Fault Tolerance Techniques

The common mode fault avoidance and fault removal techniques can increase the development cost of the program but generally do not result in an operational performance penalty. By contrast, the fault tolerance techniques can cause significant performance overheads. Therefore, not all of the techniques discussed in this section may be suitable for real time TF/TA/NOE application. Although it is difficult to quantify the overheads without a specific system design, one can separate the techniques qualitatively in low, medium, and high penalty groups.

Low overhead fault detection techniques include watchdog timers, hardware exceptions, and presence test since they require execution of zero to a few instructions at infrequent intervals. Medium overhead techniques include memory management unit if the MMU does not add significant number of wait states to memory accesses. Ada run time checks can potentially result in significant performance penalty and is an example of a high overhead technique. Finally, acceptance tests can be written to be anywhere from extremely simple such as a rate or a range check on an output variable to an elaborate program that duplicates the complete functionality of the program being checked. Thus acceptance tests can be low, medium or high overhead techniques depending upon their complexity.

Most recovery techniques do not add overheads under nominal, non-faulty operational conditions. The criterion here is the time it takes to recover from a fault since the TA/TF/NOE application tasks cannot be suspended for a very long time. The time to recover increases as the level of recovery increases. Thus, exception handlers generally require the least amount of time, task restart would require a little more followed by virtual group restart and system restart.

These are only general qualitative observations. Whether or not any of these techniques will be applicable to AFTA will depend on the specific design parameters to be determined during the detailed design phase of AFTA.

8.4.3.4. Common Mode Fault Examples

This section describes some of the common mode faults that have been observed over the years in synchronous redundant Byzantine resilient computing systems at Draper. What one observes in real time is the effect of the fault, i.e., the error symptom. Manifestations of common mode faults are polygenic. A given error symptom can be caused by

several different CMFs. For example, all members of an FTPP virtual group can go out of synchronization for a variety of different reasons such as EMI, frame overrun, etc. Common mode faults are also polysymptomatic. A given CMF can result in different error symptoms under different conditions. For example, EMI can cause a task in all members of a virtual group to produce incorrect results or it may cause all members to go out of synchronization. It is therefore easier to list the observed error symptoms than the causal CMFs. Table 8-2 lists some of the commonly observed error symptoms, their possible causes and some plausible means of detection and recovery. It should be emphasized here that the list is exemplary in nature and is not meant to be exhaustive.

ERROR SYMPTOM	POSSIBLE CAUSE	DETECTION AND IDENTIFICATION	RECOVERY TECHNIQUE (A,R,T)
(1) All channels go out of sync simultaneously or nearly simultaneously	(a) Design fault (h/w or s/w); (b) Power transient, lightning, EMI	Fast FDIR - Presence test	T: FDIR must decide when (and which) lone channel can continue running the system; recovery based on single source input rather than voted.
(2) Task gets in an infinite loop or takes an excessively long time	Software logic error (in that task)	Software watchdog timer (part of runtime)	T: Action specified by timing exception handler for the process
(3) Missed deadline by application task (i.e., task doesn't complete in time)	(a) That task or some other task takes too long; (b) Interval timer doesn't go off because of software error in setting it	Testing and validation	A: Correct the software error(s)
(4) Missed execution of tasks	(a) Problem with some higher-level task; (b) Interval timer doesn't go off because of software error in setting it	Testing and validation	A: Correct the software error(s)
(5) All channels stop	(a) Software logic error that causes it to execute STOP instr. (b) Referencing non-existent addr that doesn't return DTACK	(a) A priv violation could occur or an interrupt could come in (b) If neither of these, or no DTACK returned, watchdog causes reset after a fixed time	(a)T: Priv vio - see h/w exception below. If rupt occurs, processing could resume after erroneous instruction. (b)T: See watchdog reset below.
(6) Hardware exception on all channels	Software logic error	Hardware exception logic	T: Take action specified by hardware exception handler for the process

Table 8-2. Commonly Observed Error Symptoms of Common Mode Faults

ERROR SYMPTOM	POSSIBLE CAUSE	DETECTION AND IDENTIFICATION	RECOVERY TECHNIQUE (A,R,T)
(7) Ada software exception	Software logic error either in current task or lower level task	Ada runtime checks and task exception handlers	T: Take action specified by exception handler
(8) Channels stay in sync but produce identical incorrect output, e.g., an application task generating an invalid actuator command	Software logic error	Ada runtime checks and task exception handlers, Acceptance Tests	T: Take action specified by exception handler
(9) Channels stay in sync but produce identical incorrect output, e.g., erroneously declaring components faulty (perhaps eventually causing shutdown of a subsystem because majority of components are declared failed)	Software logic error	Testing and validation or N-Version software	R: Correct the software error
(10) Not detecting a failure (redundancy management software)	Software logic error	Testing and validation or N-Version software	R: Correct the software error
(11) Watchdog timer resets all processors	(a) Infinite loop in FDIR or runtime (i.e., when running with interrupts off) (c) No DTACK returned (c) Interval timer doesn't go off due to software error	Processors will reset; Monitor-Interlock will indicate that reset occurred because of watchdog timer	T: At restart, GPC_Init will execute procedure specified by exception handler for the current task, rather than complete re-initialization
(12) A task stops	(a) Software exception occurred for which there was no handler (b) Task Control Block written over	(a) Testing and validation (b) See below	R: Correct the software error
(13) Task destroys data, either its own or that of another task	(a) Software circumvents Ada type checks or stack limit checks (b) Process gets into infinite loop	<ul style="list-style-type: none"> Ada type checking Explicit checking when Ada type checking is not invoked Memory Management Unit 68000 User vs. Supr. modes 	T: Take exception specified by exception handler (invoked by Ada type checking)

Table 8-2. Commonly Observed Error Symptoms of Common Mode Faults (Cont.)

8.5. Effectiveness of Common Mode Fault Avoidance, Fault Removal, Fault Tolerance Techniques

There are several ways of evaluating the effectiveness of common mode fault avoidance/removal/tolerance techniques. One of the simpler ways is to pair each technique with

the CMF source against which it is effective. Qualitatively, transient (external) CMFs and permanent (external) CMFs can be avoided by proper shielding, packaging, thermal management and conservative design practices (9, 10). All other fault avoidance techniques (1-8) discussed in Section 8.4.1 will be effective against intermittent (design) faults and (permanent) design faults. All of the fault removal techniques described in Section 8.4.2 should be effective in finding design faults. The fault tolerance techniques described in Section 8.4.3 should be effective against intermittent design faults. Additionally, all fault tolerance techniques should be able to tolerate transient common mode faults. None of the fault tolerance techniques can tolerate permanent design faults or permanent external CMFs. Table 8-3 summarizes these relationships.

	CMFA		CMFR	CMFT
	1-8	9,10		
Transient (EXT) CMF		X		X
Permanent (EXT) CMF		X		
Intermittent (Design) CMF	X		X	X
(Permanent) Design CMF	X		X	

Table 8-3. Effectiveness of CMF A/R/T Techniques

Qualitative effectiveness criteria are important but do not provide the information necessary to determine which techniques one must pursue for the AFTA program. Quantitative measures would be valuable for this purpose. Unfortunately, the mechanics of how most common mode faults are introduced is not understood well enough to quantify the fraction of faults a given technique will be able to prevent, remove or tolerate. There are a few exceptions to this. For example, one can design a shield of appropriate thickness to prevent SEU upsets due to high energy particles of a given intensity or interference from HERF of a specified energy. However, such quantitative data is not available for most design faults or the techniques to avoid, remove and tolerate such faults.

At this point, then, only experience, anecdotal evidence and qualitative and subjective arguments can be used to decide on the relative effectiveness of a particular technique against a given source of common mode faults.

8.6. Suitability of Common Mode Fault Avoidance, Fault Removal, Fault Tolerance Techniques for AFTA

In order to determine the suitability of the techniques to deal with common mode faults for AFTA, it is helpful to divide the AFTA computer system into a hierarchy of elements as follows.

1. AFTA Computer System
 - 1.1 Hardware
 - 1.2 Software
 - 1.3 Power
 - 1.4 Algorithms
 - 1.1.1 Processor Element (CPU, FPU, MMU, Memory, Bus Interface, NE Interface)
 - 1.1.2 I/O Element (CPU, Memory, Bus Interface, I/O Interface)
 - 1.1.3 Network Element (Scoreboard, Global Controller, Voter, Fault Tolerant Clock, Bus Interface, NE Interface)
 - 1.1.4 Monitor-Interlock (Watchdog, Voter, Output Enabler)
 - 1.2.1 Ada Run Time System
 - 1.2.2 Core FDIR
 - 1.2.3 I/O Services
 - 1.2.4 Intercomputer Services
 - 1.2.5 Applications Software
- 1.3.1 A/C Power Source
- 1.3.2 FCR Power Source
- 1.3.3 Monitor-Interlock Power Source
- 1.4.1 OMI
- 1.4.2 Clock Synchronization
- 1.4.3 Syndrome Analysis

Once the hierarchy has been developed to a sufficient depth, one can make a 2-dimensional matrix where one dimension is the lowest level AFTA element and the other dimension is the CMF A/R/T technique. We would then choose to apply certain techniques to

certain AFTA elements based on the effectiveness criteria discussed in Section 8.5 as well as the following additional criteria.

1. Cost
2. Schedule
3. Maturity of technique/tool
4. Added Complexity

Since the AFTA program is an engineering endeavor constrained by a fixed budget and schedule, one needs to choose the techniques which are mature, timely and within the resource constraints of the AFTA program. An additional criterion is the extra complexity added by the technique. If the added complexity introduces more design errors than the use of the technique avoids or removes, then it would be a self-defeating exercise. Of course, the lack of firm quantitative data on the effectiveness of the techniques and the added complexity makes these decisions more subjective than objective.

The following discussion applies to the long term AFTA development program outlined in Section 1. The time-span covered here includes conceptual study phase, dem/val phase, FSD phase, operational phase and P³I. A subset of the techniques will be selected in co-operation with the Army and NASA for demonstration on the AFTA brassboard.

1. AFTA Computer System

Table 8-4 summarizes the choice of CMF A/R/T techniques for each element of the AFTA hierarchy based on these criteria. The numbers and letters in the table refer to the techniques discussed in Sections 8.4 - 8.6. At the system level (1.0 AFTA) the AFTA Architecture is designed to avoid CM faults (fault avoidance technique 5). For example, the Byzantine Resilient Virtual Circuit (BRVC) abstraction embodied in the FTTP hides the complexities of inter-processor communication in the presence of faults from the applications software. Any extensions to the architecture proposed during the brassboard, FSD and subsequent program phases will have to pass the test of not violating the BRVC and other complexity-reducing architectural attributes before they can be implemented.

The table also indicates that Design Reviews and Testing (fault removal techniques 1 and 3) will be used to remove CM faults at level 1.0. The asterisk implies that FR techniques 1 and 3 will be used at all levels of hierarchy below that level as well. At the system level, the only recovery technique is d (restart the whole system and establish a new system state with current sensor inputs).

2. Hardware

All of AFTA hardware elements will be designed and/or procured in accordance with MIL-STD 883C and conservative design practices will be followed (FA technique 9) and appropriate shielding, packaging and thermal management techniques (10) will be utilized.

The Processor Element and the I/O Element are Non-Development Items in the AFTA architecture. Mature Components (3) that comply with military or de facto commercial standards (7) will be procured for the PE and IOE. Additionally, formally verified PEs (2), if available in the AFTA FSD time-frame, can be used at least for the AFTA hard core functions. The hard core functions include the redundancy management tasks and the safety-critical flight control tasks. Using formally verified PEs for all the AFTA functions may not be practical due to their limited throughput in comparison with mature but formally unverified PEs.

Watchdog timers and hardware exceptions (a and b) will be used to detect CM faults in real time in the PE. For real time recovery, redundant PEs will be resynchronized (c), if necessary.

Most of the Network Element will be designed using Design Automation Tools (4). In particular, the Scoreboard, the Global Controller and the Voter and Fault Tolerant Clock will be described at least at the behavioral level using VHDL. This description may also be carried to the structural level. Formal methods (1) should also be applied to the NE hardware design. All five major blocks of the NE should be formally specified at the abstract finite state machine level. Formal verification should be carried down through the detailed hardware design to the Register Transfer Level (RTL).

In the case of ASICs, the logic synthesis from VHDL descriptions (structural and behavioral) will also utilize design automation tools. Candidates include LSI Logic's Silicon 1076 tool suite and Autologic and GDT Silicon compilers.

A software simulation (2) of the NE will be constructed. The primary purpose of the NE simulator is to provide to AFTA system software developers a substitute for the NE hardware until such time as the hardware becomes available. However, the simulator can also be used to verify the functionality of the NE design. This can be accomplished by comparing the NE simulator's response to the AFTA system software to the virtual programming model of the NE specified by NE designers.

Hardware fault injection (4) will be used in all parts of the NE hardware to uncover CM faults.

To detect CM faults in real time in the NE, the presence test (f) will be employed. This would detect a loss of synchronization of NEs. To recover from this situation, a reset will be asserted by the processors using the fiber optic links which should force NEs to resynchronize.

The Processor Bus Interface of the NE will be designed using mature components (3) as will the NE-NE interface.

The Monitor-Interlock will be subjected to hardware fault injections (4) to uncover CM faults.

3. Software

Formal methods (1) should be used to specify selected parts of the AFTA system software and applications software. A candidate language for formal specification of software requirements is Z [Spi89]. Selected parts of the software should also be formally verified. A candidate tool for verifying the correctness of Ada software is Penelope [Hir90]. Penelope is an interactive system that accepts programs from a subset of Ada and formal specifications for them. It generates verification conditions which are statements in first-order logic. Proof of these statements implies that the program satisfies its specifications.

The AFTA software will use the DoD specified standards (7) such as the programming language (Mil-Std 1815a, i.e., Ada). Good software engineering practices (8) will also be followed in the development of AFTA software.

All of the system software will be subjected to hardware faults as well as data errors and memory mutations (4). Ada run time checks and MMU (c and d) will be used to detect CM fault occurrences in real time.

The Ada Run Time System will be designed around a mature Ada-compiler-vendor supplied RTS (3). Architectural attributes (5) will be used to simplify the design of FDIR, I/O, Intercomputer services and applications software.

Additionally, for the applications software, the two candidate approaches to avoiding CM faults are the use of design automation tools (4) such as the Draper CASE tool, called IDEA, and the use of design diversity (6) to produce multiple versions from a given set of

specifications. Acceptance tests can be used to check the reasonableness of the outputs produced by the RTS and the applications software in real time to detect CM faults. In the case of CM faults in a single applications task, exception handlers (a) can try to recover from an abnormal condition. Also, the task can be purged and restarted (b) with fresh inputs in the next frame.

As far as CM faults in the power supplies and power distribution system are concerned, the only recourse is to effect a complete system restart (d). This may be a cold restart if the system state was not saved. Alternatively, one could provide low voltage detectors which will force an orderly shut-down of the system, saving the current system state in non-volatile memory. In this case, a warm restart can be effected when the power comes back on-line. Low voltage detectors are usually integrated on PEs and cause an interrupt that can be used to trigger the orderly system shut-down (see Section 4).

Finally, the algorithmic elements of AFTA such as OM1, clock synchronization, and syndrome analysis are best suited for verification by formal methods (1).

AFTA Element	Fault Avoidance	Fault Removal	Fault Detection	Fault Recovery
1.0 AFTA	5	1*,3*		d
1.1 H/W	9*,10*			
1.1.1 PE	2,3,7		a,b	c
1.1.2 IOE	3,7			
1.1.3 NE	1	2*,4*	f*	c*
1.1.3.1 SB	4	5		
1.1.3.2 GC	4	5		
1.1.3.3 V/FTC	4	5		
1.1.3.4 BI	3			
1.1.3.5 NE Int.	3			
1.1.4 MI		4		
1.2 S/W	1*,7*,8*	4*	c*,d*	
1.2.1 RTS	3		e	
1.2.2 FDIR	5			
1.2.3 I/OS	5			
1.2.4 ICS	5			
1.2.5 Appl. SW	4,5,6		e	a,b
1.3 Power				d
1.4 Algorithms	1*			
1.4.1 OM1				
1.4.2 Clock Synch.				
1.4.3 Syndrome Analysis				

Table 8-4. Application of CMF A/R/T Techniques to AFTA

c-4

8.7. Plan for Implementation of CMF Avoidance, Removal, Tolerance Techniques

To deploy a fault tolerant computer system that is resilient to common mode faults as well, the planning must begin at the earliest phase of the program and the plan must be carried out through development and deployment of the product. It is appropriate to begin by defining a plan of action in the conceptual study phase. Steps leading to the definition of the plan of action have included the following activities. Sources of common mode faults in AFTA have been identified. A three-pronged approach to make AFTA CMF-resilient, consisting of fault avoidance, fault removal, and fault tolerance, has been developed. Techniques and tools for each of the three prongs have been enumerated. Each of the three prongs has been matched to the type(s) of CMFs against which it is effective. This section now outlines the time-line for using various tools and techniques and other actions that will be required throughout the AFTA life-cycle to make AFTA CMF-resilient.

8.7.1. Demonstration/Validation Phase

This phase of the AFTA development will last 36 months. Activities during dem/val include AFTA detailed design, brassboard fabrication, coding, and integration, demonstration of brassboard with an application and AFTA validation. Demonstration and validation activities will be carried out initially at Draper and subsequently at Army AVRADA and also possibly at NASA Langley Research Center.

If CMF-resilience is a serious goal of the AFTA project, then enough funding should be found to support the following CMF A/R/T activities during the dem/val phase. (In practice, the FSD funding levels are typically much higher than the dem/val funding levels. This may necessitate postponing some of the activities from the dem/val to the FSD phase.)

8.7.1.1. AFTA System

Any extensions/modifications to the AFTA computer system should be examined for compliance with the attributes that help reduce, manage and hide the system complexity (see Section 8.4.1.5).

8.7.1.2. Hardware Design

The Network Element design should be described at the behavioral and at the structural level in VHDL. The NE hardware should be designed in accordance with Mil-Std 883B/C for the full temperature range. Conservative design practices should be followed.

Features that will help detect and tolerate CMFs should be designed into the NE. Examples include a watchdog timer to reset the NE when it goes into an undesired state and time-outs for PE input/output buffer full conditions.

Formal methods should be used to begin the verification process of the NE hardware design. As a first step, an abstract finite state machine-level specification of the three major blocks of the NE that are bus-interface-independent, i.e., the Scoreboard, the Global Controller, and the Voter/Fault Tolerant Clock should be constructed.

Rigorous Preliminary and Critical Design Reviews (PDR and CDR) of the NE design by peers, superiors, and government contract monitors should be carried out.

8.7.1.3. Software Design

The Dem/Val software should be designed using good software engineering practices outlined in Section 8.4.1.8. A waterfall software development methodology, starting with software requirements specifications and ending with detailed design, should be followed.

Design automation tools such as Draper's CASE, HTI's 001, Cadre's Teamwork should be examined for applicability to AFTA software design and coding.

Selected subset of AFTA system software should be formally specified in the Z language or a suitable software specification language. Selected subset of AFTA system software should be formally verified using Penelope or a suitable software formal verification tool.

The software should be coded in Mil-Std 1815a Ada language. The Run Time System should be based on the XD-Ada-supplied and Draper-modified RTS that is currently being used in the US Navy's SSN-21 Seawolf Ship Control Computer and has also been ported to the FTTP Cluster C2+.

System software developed under AIPS, Seawolf and FTTP projects such as I/O System Services and Inter-Processor Communications Services should be examined for AFTA

reuse. CAMP software libraries should be acquired and examined for applicability to AFTA.

CMF detection and recovery mechanisms such as those discussed in Sections 8.4.3.1 and 8.4.3.2, respectively, should be designed into the AFTA operating system. These techniques should be sufficiently broad to cover the commonly observed error symptoms of CMFs presented in Table 8-2.

Rigorous Preliminary and Critical Design Reviews (PDR and CDR) of the software design by peers, superiors, and government contract monitors should be carried out.

8.7.1.4. Hardware-Software Test and Integration

The primary emphasis during the integration phase will be on fault removal techniques. The three major techniques to be used during this phase are testing, fault/error injection, and discrepancy reporting.

Software testing should follow the unit, module, and functional testing paradigm. The major complexities of the AFTA architecture are in the dimensions of hardware redundancy and parallel processing. The AFTA architecture hides the redundancy dimension from most of the software by providing a Byzantine Resilient Virtual Circuit abstraction. This will allow all of the system software, except FDIR, to be tested in a non-redundant environment. The parallel processing dimension is also hidden from applications software by the inter-processor communication services. The testing of most AFTA software can therefore utilize the tools and techniques developed for conventional computer architectures. For the FDIR, inter-processor communications services and possibly some I/O services, it will be necessary to develop a more sophisticated debugging environment. It is essential to provide software developers debugging tools that gives them visibility into the workings of the parallel-redundant computer. Development of such a debugging environment should be a priority of the AFTA dem/val phase.

Hardware testing during dem/val mainly pertains to the NE which can be tested using traditional hardware testing techniques if the design does not contain any ASICs. The NE simulator should be used to verify the NE functionality against the NE virtual programming model specified by the hardware designers. The design of the NE hardware, in turn, should be verified against the NE simulator.

Discrepancy reports should be filed starting with the testing phase. This activity should continue through various stages of hardware-software integration. As design and manufacturing errors are discovered and fixed, tests should be rerun to duplicate the discrepancies.

Once the system has been integrated, it should be subjected to extensive fault and error injections with and without applications code executing. Initial testing will be with faults/errors in a single fault containment region and only in the NE. System parameters of interest such as fault detection, identification and recovery times should be recorded. Cause of any faults not detected, misdiagnosed, or improperly recovered from should be identified. Impact on performance should be evaluated to ascertain that no scheduling dead-lines were missed due to fault handling transients in the system performance and no unexpected data corruption occurred. Subsequently, common mode faults should be injected to test AFTA's ability to tolerate CMFs.

8.7.2. Full Scale Development Phase

The CMF A/R/T activities during the FSD phase parallel those during the dem/val phase. However, they will be on a much larger scale.

For example, the formal specifications of software, applied to a small subset of software during the dem/val phase, should be carried to as much software as possible. The formal description of the NE at the finite state machine done during the dem/val phase should be carried down to the gate level. Formally verified PEs should be examined for inclusion in the FSD hardware demonstration.

The CASE tools for software design and development should be applied more extensively across a broader spectrum of system and applications software. The software development should rigorously follow Mil-Spec-2167a. Extensive simulations of the NE ASICs designs should be carried out. Multiple versions of flight critical application code should be developed.

More extensive fault detection and recovery techniques should be designed into the AFTA system software, the NE and the PEs.

Same fault removal techniques as those in the dem/val phase will be applied here as well. However, their application should be much more extensive. The testing should be over a wider input range. The DRs should be logged with enough information to construct

the software reliability growth models. The fault/error injection should be expanded to cover the PEs and the IOEs as well as the NE.

A closed loop testing of the complete AFTA system including the computer and I/O devices tied to a dynamic simulation of the helicopter/ground vehicle should be carried out under normal conditions, with faults, and under maximum application load and faults.

The hardware should also be subjected to various Environmental Screening and Stress (ESS) tests. Any component failures should be examined for design errors and corrected.

8.7.3. Production Phase

The major sources of common mode faults in the production phase are the manufacturing defects. Appropriate quality control measures are necessary to ascertain that the AFTA systems are manufactured in accordance with design specifications. It is also very important to track the changes in the system requirements that affect the AFTA design very carefully. If design changes become unavoidable due to changing requirements, it would be necessary to go through the critical steps of the previous two phases to make sure that the design changes do not introduce new errors.

8.7.4. Deployment Phase

When AFTA systems are deployed in the field, data on faults, errors, failures, and anomalous behavior should be collected and analyzed. The cause of each event should be examined and categorized as a random hardware fault or a CMF or a potential CMF. Sometimes a design error can masquerade as a random hardware fault as, for example, when it causes only a single fault containment region to fail. Therefore, it is important to do the cause and effect analysis for each observed event. This feedback can then be used to remove the source of the CMF. The feedback process should also be used to examine the efficacy of the three-pronged approach, i.e., CMF avoidance/removal/tolerance techniques. Every CMF that slips through this process should be examined to determine the effectiveness and the weaknesses of various techniques in avoiding, removing, and tolerating CMFs. The field data should be studied with the goal of improving the CMF A/R/T approach.

8.7.5. Pre-Planned Product Improvement Phase

Ideally, all of the CMF A/R/T techniques suggested in this report should be applied to the development of AFTA during the dem/val and the FSD phase. However, it may turn out that the AFTA schedule and the availability of the tools and techniques do not coincide. In that case, some of the activities can be deferred for the Pre-Planned Product Improvement (P³I) phase. For example, if a formally verified microprocessor is not available in time to meet the FSD schedule, it can be added later on to the AFTA as a part of the P³I activity.

This page intentionally left blank.

9. Analytical Models

This section describes the quantitative models are to be used in analytically evaluating AFTA. Quantitative models are presented for effective throughput, effective intertask communication bandwidth and latency, effective input/output bandwidth and latency, reliability and availability under two typical AFTA redundancy management policies, weight, power, volume, and life cycle cost.

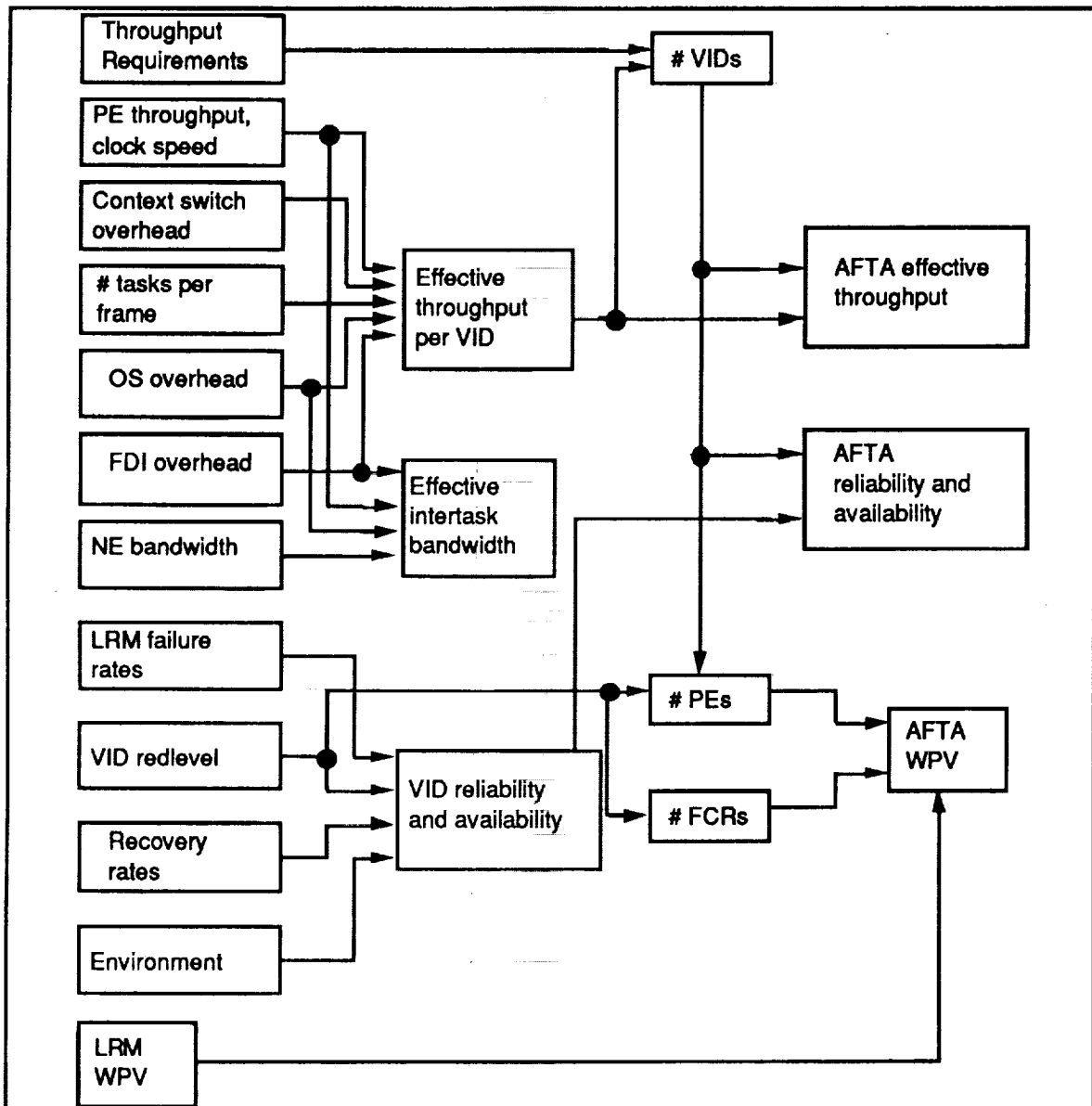


Figure 9-1. AFTA Methodology Information Flow

The inputs to these models come from MIL-HDBK-217E failure rate data, empirical test and evaluation, and other sources. The relationship among the analytical models, the model inputs, the AFTA configurable parameters, and the AFTA requirements, is depicted in Figure 9-1.

9.1. Performance Model

9.1.1. Delivered Throughput

In the initial stages of architecture synthesis only overall throughput requirements are available, while in the early stages of development the effective throughput of a given architecture can in turn only be roughly estimated. A rough delivered throughput model is suitable for this situation. In this model, one begins with the raw throughput, expressed in suitable terms such as DAIS MIPS. Denote this quantity $X_{VG, raw}$. The VG throughput estimate is reduced to a value denoted $X_{VG, delivered}$ by overheads such as the rate group (RG) dispatcher, synchronization delays, RM overheads, contention effects, and context switches.

The effective throughput available to an application running on a parallel processor is a strong function of the efficiency of the mapping of the application task to the parallel processing resources, and is impossible to plausibly generalize. Therefore in the current report we will calculate the delivered throughput of an AFTA simply as the sum of the throughputs of its constituent VGs. Thus, in an AFTA configuration consisting of N_{VG} VGs each having throughput $X_{VGi, delivered}$, the delivered throughput is

$$X_{AFTA, delivered} = N_{VG} X_{VG, delivered} \quad (9.1)$$

Estimation of a VG's delivered throughput requires a frame-by-frame analysis of the AFTA OS and redundancy management (RM) overheads. For convenience, a diagram of the RG frames used by the AFTA scheduler is repeated below.

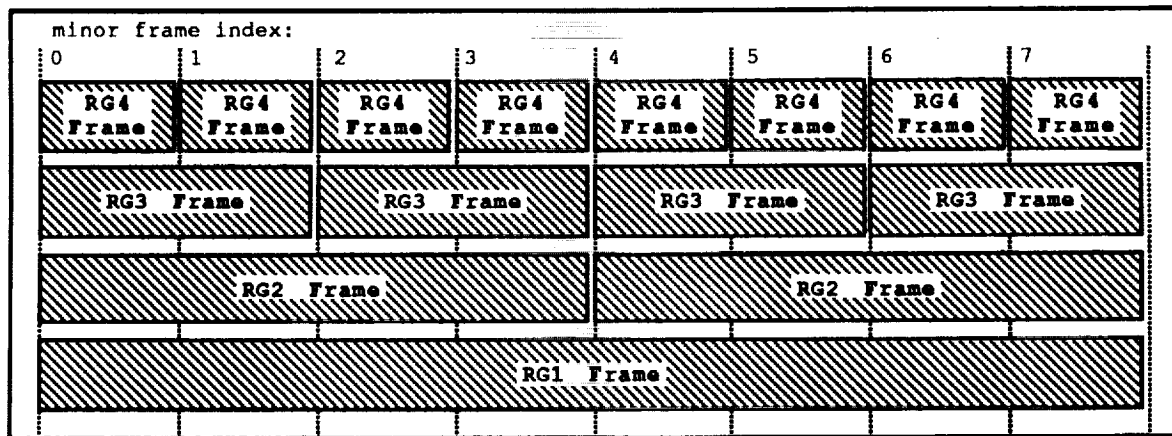


Figure 9-2. Mapping of RG Frames to Minor Frames

The overall approach to determining AFTA OS and fault tolerance-related overhead, and hence delivered throughput, is to calculate and verify the time required to perform these functions. Time is used instead of a parameter such as the number of instructions required to execute a particular overhead function because time is a directly measurable and therefore verifiable quantity, whereas instruction counts are notoriously misleading when used to estimate performance. In addition, many of the overheads include operations to which an instruction count and processor throughput are irrelevant, such as accessing the network element. As the detailed design of AFTA proceeds, the overhead time estimates will be refined and correlated closely to parameters such as processor instruction execution rate and Network Element bandwidth. However, it is worth noting that, even at validation and verification time, the execution times are primary measured parameters.

At each minor frame boundary, a particular set of RGs have completed their iterations and are ready to initiate a new iteration. These RG sets as a function of the frame boundary are shown in Table 9-1.

Frame Boundary	Completed RGs	Started RGs
7-0	4, 3, 2, 1	4, 3, 2, 1
0-1	4	4
1-2	4, 3	4, 3
2-3	4	4
3-4	4, 3, 2	4, 3, 2
4-5	4	4
5-6	4, 3	4, 3
6-7	4	4

Table 9-1. Completed/Started RGs vs. Frame Boundary

At each minor frame boundary several functions are executed which contribute to the OS overhead[†]. The chime interrupt handler synchronizes the VG and performs time management functions; subsequently the dispatcher performs other metabolic housekeeping. The time required to perform these functions is denoted

$$T_{HK}$$

Subsequently, the dispatcher transmits all messages emanating from RGs whose frames have just completed. The time required to perform this function depends on the number of RGs that have just completed, the number of tasks in each RG, the number of messages sent by each task, the size of each message, and contention with other VGs for the Network Elements' message passing services. The time required to perform this is denoted

$$T_{SEND, i} = \sum_{k=1}^{N_{MESSAGES, i}} T_{SU} + S_k T_p \quad (9.2)$$

where $N_{MESSAGES, i}$ = the number of messages sent in frame i , T_{SU} is the setup time required to begin sending a single message, S_k is the size (in Network Element packets) of outgoing message k , and T_p is the incremental time required to send one packet.

Next, the dispatcher updates the incoming message queues for all tasks which have received messages during the previous frame, and updates the frame markers for tasks in

[†] See Section 5.3 for a detailed description of the dispatcher functionality.

RGs which will be started on the current frame boundary. The time required for this depends on the number of RGs that have just completed, the number of tasks in each RG, the number of messages received by each task, and the size of each message. The time required to perform this is denoted

$$T_{\text{RECEIVE}, i} = \sum_{k=1}^{N_{\text{MESSAGES}, i}} T_{\text{SU}} + S_k T_p \quad (9.3)$$

where (overloading names to avoid needless notation proliferation) $N_{\text{MESSAGES}, i}$ = the number of messages received in frame i , T_{SU} is the setup time required to begin receiving a single message, S_k is the size (in Network Element packets) of incoming message k , and T_p is the incremental time required to read and process one packet from the Network Element*.

Finally, before suspending itself, the dispatcher enables execution of all tasks residing in RGs which can be started on the current frame boundary by setting an event for them; the time required for this depends on the time required to set an event, the number of RGs to be started in the next frame, and the number of tasks in each RG. The time required to perform this is denoted

$$T_{\text{EV}, i} = N_{\text{TASKS}, i} T_{\text{EV}} \quad (9.4)$$

where $N_{\text{TASKS}, i}$ = the number of tasks to be started in frame i , and T_{EV} is the time required for the dispatcher to set an event.

$T_{\text{D}, \text{major}}$, the time consumed by the dispatcher over all eight minor frames (i.e., one major frame) is estimated as:

$$T_{\text{D}, \text{major}} = \sum_{i=0}^7 (T_{\text{HK}} + T_{\text{SEND}, i} + T_{\text{RECEIVE}, i} + T_{\text{RV}, i+1}) \quad (9.5)$$

where i refers to the minor frame just completed, and is computed modulo 8.

* Strictly speaking, a recipient PE reads a packet from an NE upon reception of a packet delivery interrupt, according to the method outlined in Section 4, and this overhead is spread out within a frame. The current approach to estimating the overhead consumed by this process is to lump it all together at the frame's end.

The FDI task immediately follows the dispatcher in each minor frame. This task executes a complex suite of functions which are described in Section 5. The exact set of functions included in the FDI task is not known at the current phase of development and in fact may vary from mission to mission depending on a mission's temporal constraints. At the current level of modeling granularity, the temporal overhead due to FDI task execution in a minor frame is abstracted as $T_{FDI, \text{minor}}$, and the total temporal overhead due to FDI task execution over the major frame is therefore

$$T_{FDI, \text{major}} = 8T_{FDI, \text{minor}} \quad (9.6)$$

The context switch time incurred by the R4 tasks, the dispatcher, and the FDI tasks in the major frame is equal to the context switch time per task (T_{CS}), times the number of R4 tasks ($N_{TASKS, R4} + 2$), times the number of minor frames per major frame (8):

$$T_{CS, \text{major}, R4} = 8(N_{TASKS, R4} + 2)T_{CS} \quad (9.7)$$

For lower-frequency RGs, upper bounds for the context switch times are computed. The context switch time incurred by the R3 tasks is upper-bounded by the context switch time per task (T_{CS}), times the number of R3 tasks ($N_{TASKS, R3}$) plus the maximum number of times that RG R3 can be preempted in a major frame (4), times the number of minor frames per major frame (4):

$$T_{CS, \text{major}, R3} = (4N_{TASKS, R3} + 4)T_{CS} \quad (9.8)$$

The context switch time incurred by the R2 tasks is upper-bounded by the context switch time per task (T_{CS}), times the number of R2 tasks ($N_{TASKS, R2}$) plus the maximum number of times that RG R2 can be preempted in a major frame (6), times the number of minor frames per major frame (2):

$$T_{CS, \text{major}, R2} = (2N_{TASKS, R2} + 6)T_{CS} \quad (9.9)$$

The context switch time incurred by the R1 tasks is upper-bounded by the context switch time per task (T_{CS}), times the number of R1 tasks ($N_{TASKS, R1}$) plus the maximum number of times that RG R1 can be preempted in a major frame (7), times the number of minor frames per major frame (1):

$$T_{CS, \text{major}, R1} = (N_{TASKS, R1} + 7)T_{CS} \quad (9.10)$$

The total temporal overhead per major frame due to context switches is upper-bounded by

$$T_{CS, major} = T_{CS, major, R4} + T_{CS, major, R3} + T_{CS, major, R2} + T_{CS, major, R1} \quad (9.11)$$

The total temporal overhead per major frame due to the dispatcher task, FDI task, and context switches, is

$$T_{OVERHEAD, major} = T_{D, major} + T_{CS, major} + T_{FDI, major} \quad (9.12)$$

Let T_{major} denote the period of a major frame. Then the fractional overhead due to the dispatcher task, FDI task, and context switches, is upper bounded by

$$OH = \frac{T_{OVERHEAD, major}}{T_{major}} \quad (9.13)$$

Let $X_{VG, raw}$ denote the raw throughput of an AFTA VG. As a simple first-order engineering approximation of the VG's delivered throughput after deduction of the various overheads, one may use

$$X_{VG, delivered} = (1 - OH) X_{VG, raw} \quad (9.14)$$

9.1.2. Intertask Communication

Intertask communication is achieved in AFTA via sending and receiving messages according to the design described in Section 5. For uniformity of programming and transparency of distributed processing resources, message passing is used both for intra-VG and inter-VG communication. As described in Section 5, RG tasks may enqueue messages for subsequent transmission by the AFTA intertask communication services[†]. The time required to enqueue a message is denoted $T_{ENQUEUE MESSAGE}$ and is a parameter to be verified during the Dem/Val. Upon completion of an RG frame, the AFTA intertask communication services transmit packets emanating from the just-completed and all lower-frequency RGs to the destination VG via the Network Elements. All messages emanating from a VG are transmitted on a RG boundary, which is in turn determined by the sending VG's timer interrupt. This has the intent of minimizing the jitter and skew with which messages are transmitted by a VG. It also has the effect of delaying the transmission of a

[†] In extenuating circumstances, RG4 tasks may send messages immediately without waiting for a frame boundary because they are nonpreemptible.

task's messages until the end of its RG frame, so the message latency, as measured with respect to the moment the sending task emits the message, can be up to one RG frame, depending upon where in the RG frame the task is scheduled. This is illustrated graphically in Figure 9-3, in which each RG has one message to send, each denoted by a **boldface m**: RG4 enqueues **m1**, RG3 enqueues **m2**, RG2 enqueues **m3**, and RG1 enqueues **m4**. The messages are transmitted by the AFTA communication services at the frame boundaries. The latency incurred by message **m3** is highlighted in the figure.

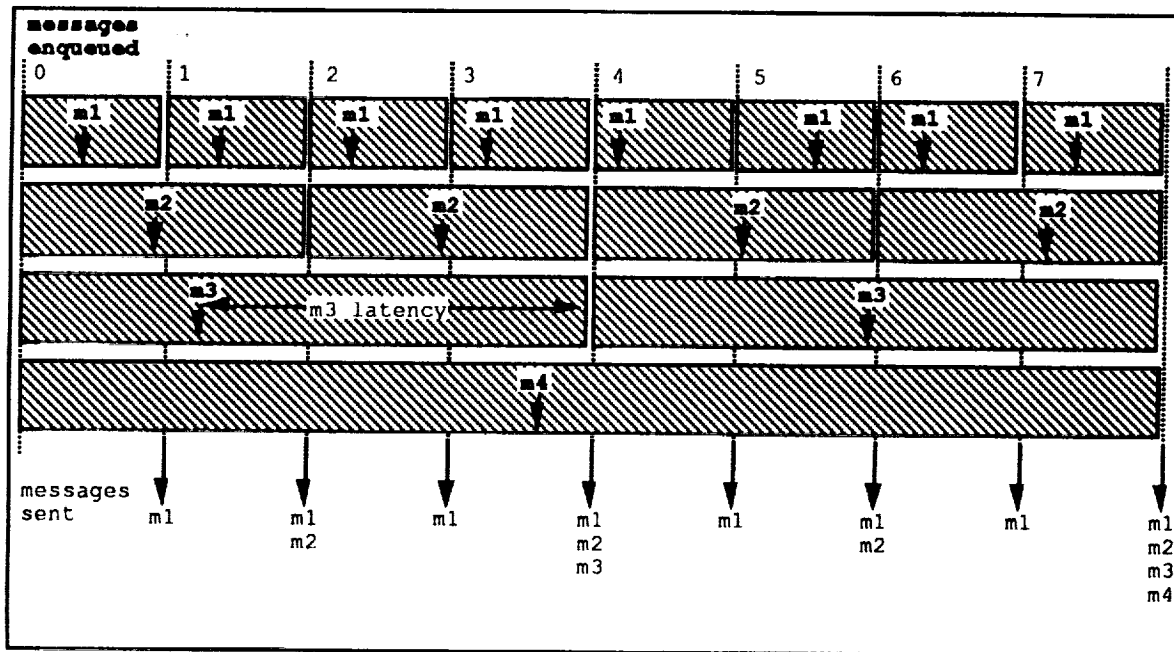


Figure 9-3. RG Message Passing

The time at which a task is scheduled in its frame and the particular frame within which it is scheduled have a significant impact on intertask communication latency and bandwidth. Let $T_{\text{latency, RG}}$ denote the time from message enqueueing (the **boldface ms** in Figure 9-3) to the task's next RG frame boundary, at which point the message is transmitted by the communication services (the plainface ms in Figure 9-3). While it must be empirically validated to obtain an accurate estimate, this parameter is largely under the control of the application designer. However, it is not recommended that $T_{\text{latency, RG}}$ be relied upon for correct execution of the application task because $T_{\text{latency, RG}}$ depends upon the relationship between the sending task's invocation of the message passing services and the frame interrupt. This in turn depends upon task execution times, which may vary widely from iteration to iteration, inducing unwanted jitter, skew, and validation difficulties. It also makes it more difficult to modify the tasking schedule when desired. It is preferable to specify tim-

ing parameters with respect to frame boundaries, which, because they are determined by highly accurate crystal oscillators on the PEs, have low skew, low variance, and are more validatable. Consequently, in the following analysis all timing parameters will be specified and verified with respect to the timer interrupt demarcating the boundary of the just-completed RG frame.

On frame boundaries, the AFTA communication services transmits all enqueued packets from completed RGs into the Network Element. The latency incurred by this function has two components. First, each outgoing packet must be written over the PE-NE bus into the Network Element; let T_{XMIT} denote this stochastic time interval, which must be paid for each packet to be transmitted. Finally, let T_{NE} denote the stochastic time interval required for the Network Element ensemble to perform the requested message transmission according to the exchange rules described in Section 4. The total time required to send a single message of size S_K packets using this procedure is

$$T_{SEND} = S_K(T_{XMIT} + T_{NE}) \quad (9.15)$$

The packet transmission procedure is illustrated in Figure 9-4.

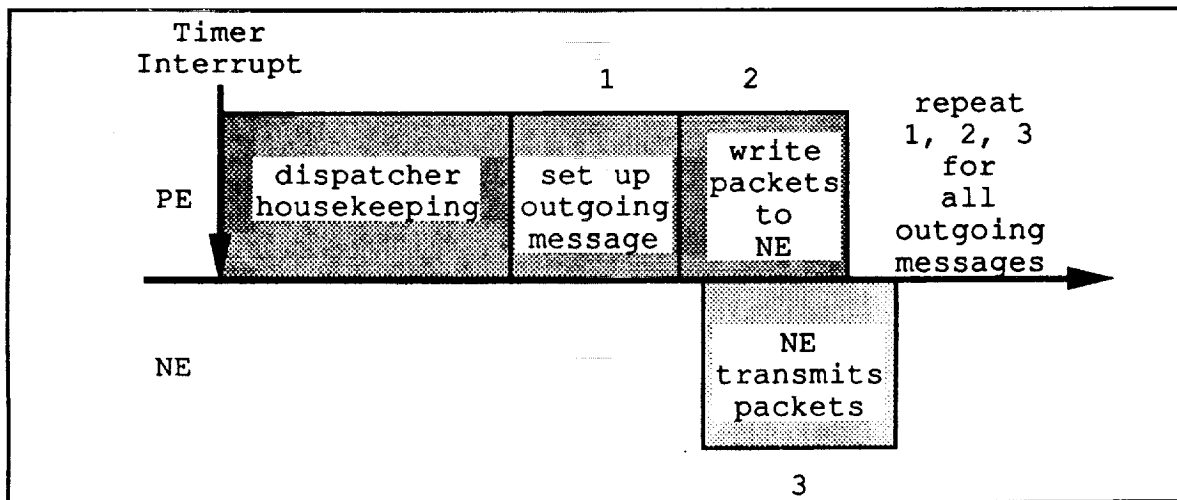


Figure 9-4. Outgoing Message Processing

Messages arrive at a destination VG at arbitrary times during a frame. The recipient PEs of the destination VG receive a packet delivery interrupt from the NE and read the packet from the NE into the PE's private memory. (The loss of throughput incurred due to this asynchronous activity is debited in the delivered throughput model.) However, the packets are not assembled and made available to destination tasks until the termination of

the RG hosting the destination task; therefore the latency (the time between message arrival and delivery to the destination task) is a function of the phase relationship between message reception and the frame boundary. Again, this phasing cannot in general be counted upon to yield a known timing relationship, so the timing specification should be performed with respect to the destination task's RG frame boundary following the reception of the last packet of a message. If some desired phase relationship must be maintained, it can be obtained via the frame phasing technique described in Section 5. Upon a frame boundary boundary, the AFTA intertask communication services process the received packet queue, constructing and delivering messages to tasks which are at an RG boundary. Note that, unless it is a latecomer, the message is already in the PE's private memory because it was read from the NE during the packet delivery interrupt service routine. The time required to perform incoming message processing is an increasing function of the number of new messages to be assembled, the number of packets received in the previous frame, and the number of new messages completed and ready for delivery to destination tasks. Denote this time $T_{\text{RECEIVE MESSAGE}}$. A plausible parametric formulation for this time interval is currently unknown and will be developed and verified during subsequent phases of the AFTA development.

2.1.3. Input/Output

The input latency is defined to be the time interval between the sampling of a physical quantity by an input device and the delivery of the digital representation of that quantity to the recipient task. The output latency is defined to be the time interval between the production of a digital quantity by a source task and the delivery of that digital quantity to the device which converts that digital quantity to a physical quantity.

As described in Sections 4 and 5, AFTA will support numerous I/O devices and controllers, including FTDB, MIL-STD-1553, mass semiconductor memory, rotating media memory, discretes, analog, RS232, Ethernet, and possibly others to be determined at a future date. The techniques used for accessing these devices have been partitioned into two classes. In the first technique, known as *concurrent I/O*, an I/O task resident on one or more members of a VG responsible for accessing an I/O device initiates an I/O transaction by writing commands and data to the device. Immediately thereafter, the VG may initiate other I/O, resume processing other I/O, or resume to other tasks while the I/O transaction completes concurrently, under control of an autonomous I/O controller. After a specified time interval, the I/O system services may return to the autonomous controller to process the input and/or status data resulting from the transaction. It is the intent that at any given

time several concurrent I/O transactions will be in progress, probably under the control of several VGs, in order to maximize AFTA I/O performance. Concurrent I/O is expected to be useful for operation of complex I/O devices such as network controllers and rotating storage media, where it is a waste of CPU time to wait around for lengthy transactions to complete. The second I/O technique available in AFTA is called *sequential I/O*, in which one or more members of a VG responsible for accessing an I/O device performs an I/O transaction and wait until the I/O activity is completed before initiating other I/O or resuming to other tasks. Sequential I/O is suitable for accessing fast, low-latency devices which may require atomic access, such as discrete input and output complexes, analog to digital converters, digital to analog converters, etc. It is simpler than concurrent I/O, but should be used with discretion since it nonpreemptively monopolizes the VG.

Prior to being provided to an output device, data may be transferred from the VG(s) hosting the source task to the VG(s) responsible for the I/O activity, or, if they are one and the same, the output data may be voted prior to being output; both such actions utilize the Network Element message passing capabilities. In addition, input data may have to be transferred from the I/O VG(s) to the destination VG(s) using one or more Class 2 exchanges. A subset of the large number of possibilities is enumerated in Section 5.

During the AFTA Conceptual Study, it was judged that generation of an I/O performance model general enough to describe the disparate I/O devices, access techniques, and input and output data distribution options possible in AFTA would be quite time-consuming. Therefore it has been decided that, for reasons of expediency, construction of the AFTA I/O performance model(s) will be deferred until more information is obtained about anticipated I/O devices; this will occur during the Detailed Design phase of the program.

9.2. Reliability and Availability Models

The reliability and availability of an AFTA implementation is a function of the number of FCRs and PEs, the VG redundancy levels, the mission environment, the operational and maintenance scenario, and fault recovery procedures. Precise mathematical definitions of the terms "reliability" and "availability" are used in this report. While high reliability and availability contribute to dependable operation, they should not be construed to exhaustively connote all attributes of dependable systems.

AFTA fault recovery options are enumerated in Section 5.6.6, and each one has a strong impact on the overall AFTA reliability and availability. Since the construction of re-

liability models for all the options described in Section 5.6.6 is beyond the scope of the Conceptual Study phase, two have been selected for evaluation as being appropriate for two extremes of temporal constraints which might be imposed on AFTA.

The first class of options, of which the *graceful degradation* and *Network Element masking* in Section 5.6.6 are examples, are appropriate for an operational mode in which little if any time is available for fault recovery. In this case, a faulty component in a redundant VG or an NE is immediately disabled upon detection, with no lengthy fault recovery attempted. No effort is made to discriminate between transient and permanent faults for the purpose of performing on-line recovery, in effect treating all faults as permanent until a more relaxed operational regime is entered. This option has the advantage of incurring no dropout of functionality, but has the disadvantage of irreversibly reducing the redundancy level of the faulted VG and hastening its demise due to redundancy exhaustion. Therefore it may be viewed as being best suited for short missions having fast real-time constraints, such as real-time control of mission-critical helicopter functions.

Figure 9-5 illustrates this fault recovery option: after the first failure of member A of quadruply-redundant VG₁, the faulted member is disabled, reducing VG₁'s redundancy level to triplex. A second failure of one of VG₁'s members, say B, reduces its redundancy level to "degraded triplex." For a degraded VG, the Network Element's main data path packet voter masks the input from the faulted member and does not include it in the vote. The Scoreboard, however, continues to consider a degraded VG's faulted channel when calculating the VG's voted Output Buffer Not Empty (known as OBNE, an indication that the VG has a packet to be transmitted from its Output Buffer) and voted Input Buffer Not Full (IBNF, an indication that the VG is capable of receiving at least one packet in its Input Buffer)[†]. This is to allow a faulted member of a degraded VG to remain in synchronization with its parent VG to facilitate recovery operations. This capability is more robust and useful for degraded quadruplex VGs than for degraded triplex VGs.

A third failure in VG₁, say of member C, reduces its redundancy level to simplex, and a fourth failure results in the loss of the functionality supported by VG₁. The probability of successfully transitioning from a faulted degraded triplex VG to a nonfaulty simplex is significantly less than unity, and is represented by the "duplex coverage," C_D .

[†] See Section 4 for a discussion of this terminology.

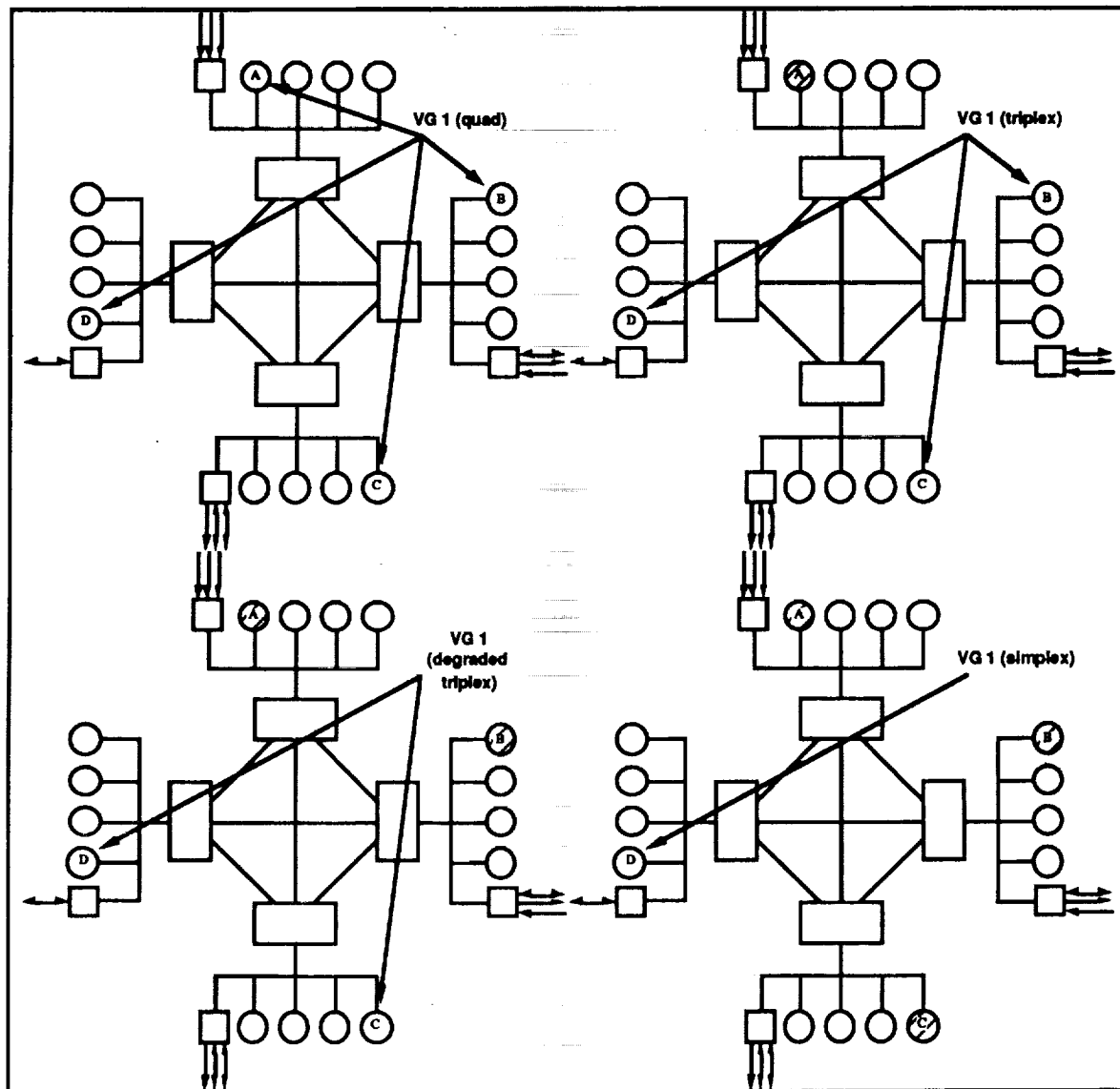


Figure 9-5. Graceful Degradation of Quadruplex VG₁

When a fault recovery time on the order of a second or two is permissible, a wider range of fault recovery options are available. Representatives of this class of options are listed in Section 5.6.6 as *processor resynchronization*, *processor reintegration*, *processor replacement*, *processor replacement with initialization*, *task migration*, and *Network Element resynchronization*. All of these recovery options are characterized by their capability to seek and find components sufficient to maximize the likelihood of forming a desired configuration of redundant VGs, followed by either initializing or copying the state of the newly reintegrated component into agreement with the surviving members of the faulted VG. As is mentioned earlier, this process, while maximizing the effective use of the re-

configurable AFTA components, consumes one to two seconds to perform. As an example of such a strategy in the context of the previous example, we reconsider the case of a processor replacement fault recovery option applied to VG₁[†]. After a failure of member A of VG₁, VG₁'s redundancy level can be restored by switching in (say) the PE adjacent to member A. After the second failure of member B, a spare processor may be reintegrated, again restoring VG₁'s quadruplex redundancy level, and so on and so forth (Figure 9-6). This can continue until all the spares allocated to repairing VG₁ are exhausted, at which point the VG₁ fault recovery policy may revert to the graceful degradation policy described above, or another policy may go into effect.

The more leisurely fault recovery options in this class are more suited to less stressful real-time operational regimes and missions, such as during the hiatus phase of the flight mission where availability is to be maximized, or during a long ground mission where one or two second dropouts are a reasonable tradeoff for significant mission longevity enhancement.

[†] Different VGs may have different fault recovery options, and the same VG's fault recovery option can vary over the course of a mission.

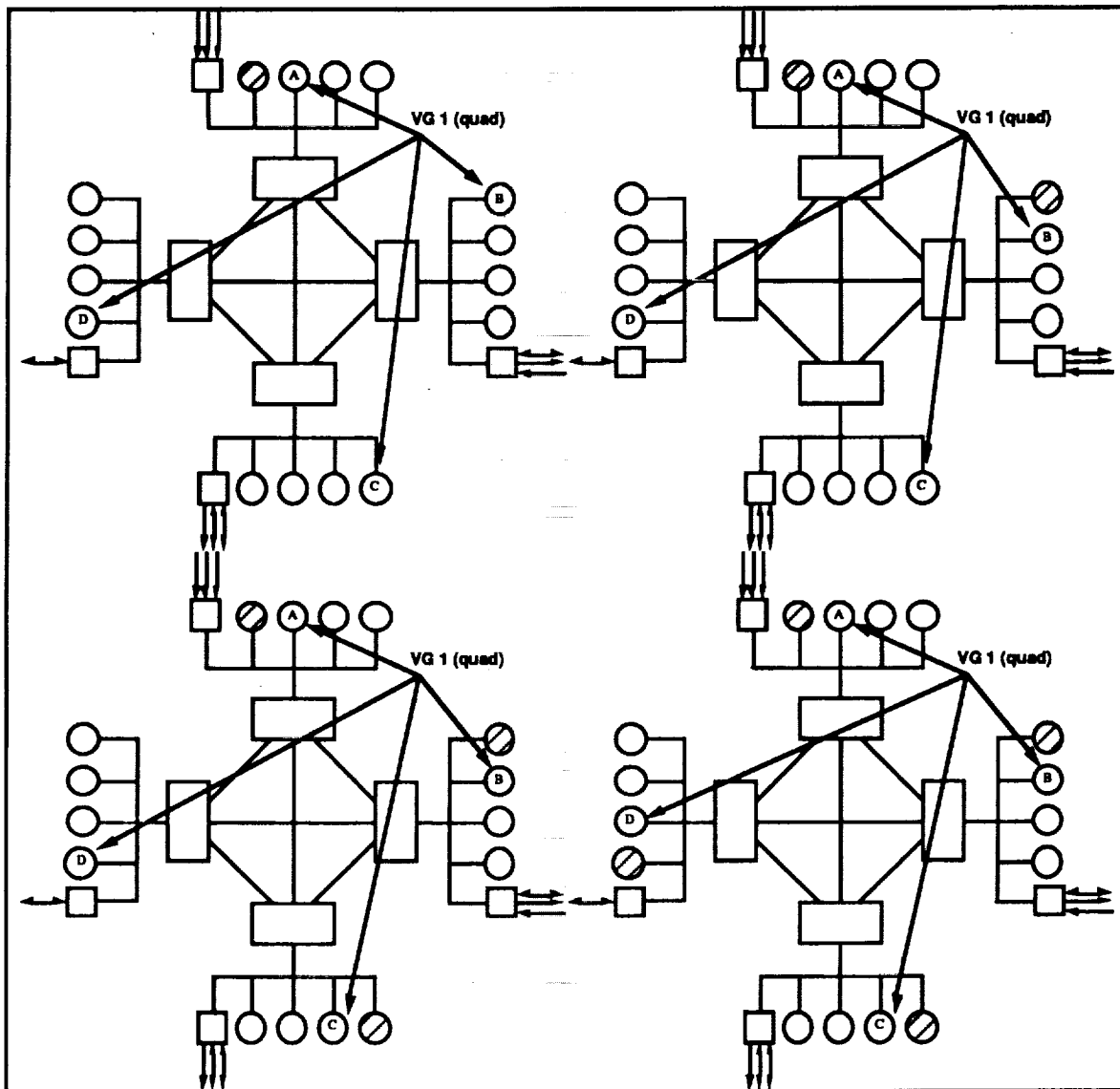


Figure 9-6. Processor Replacement Redundancy Management for Quadruplex VG₁

The following sections present formulations of the probability that AFTA can perform its intended functions, i.e., form the requisite number of functioning VGs, when managed according to the two fault recovery policies outlined above. Depending upon the use to which AFTA is put at a given time, a given formulation will be equivalent to either “reliability” or “availability”. For example, when the processor replacement strategy is used during hiatus to maximize AFTA availability, the formulation will refer to “AFTA Mission Availability”, whereas when it is used to calculate the probability that the Fault Tolerant Navigation Processor is capable of performing its intended function during a mis-

sion, the same formulation yields the “FTNP Mission Reliability”. To attempt to generalize the meaning of the formulations, the following notation is adopted.

The probability that AFTA can perform its intended functions, i.e., form the requisite number of functioning VGs, when managed according to the graceful degradation class of redundancy management policies is denoted

PGD

The probability that AFTA can perform its intended functions, i.e., form the requisite number of functioning VGs, when managed according to the processor replacement class of redundancy management policies is denoted

PPR

Recall from Section 2 that the reliability of the system is equal to the probability that all functions needed to execute the mission are operational, or

$$R_{sys} = \text{Prob}(F_j \text{ operational}, \forall F_j \in S) \quad (9.16)$$

The Function Reliability is the probability that a given function F_j can be executed because its resources are operational

$$R_{Fj} = \text{Prob}(\text{resource}_i \text{ operational}, \forall \text{resource}_i \in F_j) \quad (9.17)$$

The System Reliability is then

$$R_{sys} = \text{Prob}(\text{resource}_i \text{ operational}, \forall \text{resource}_i \in F_j, \forall F_j \in S) \quad (9.18)$$

The probability that all needed VGs are functional is

$$R_{sys} = \text{Prob}(VG_i \text{ operational}, \forall VG_i \in F_j, \forall F_j \in S) \quad (9.19)$$

If all VG reliabilities were independent, this would reduce to

$$R_{sys} = \prod_{VG_i \in F_j, F_j \in S} R(VG_i) \quad (9.20)$$

Unfortunately they are not: they are correlated through their joint dependence upon failure of FCRs in which they have common members. However, if conditioned upon the

failure of the Fault Containment Regions in which their members reside, the probabilities for VG reliability do become independent. For a single VG, one may write

$$\begin{aligned}
 R(VG_i) &= R(VG_i | \text{no FCR faults}) \Pr(\text{no FCR faults}) \\
 &+ \sum_{j=1}^{NNEs} R(VG_i | FCR_j \text{ faulty}) \Pr(FCR_j \text{ faulty}) \\
 &+ K \sum_{j=1}^{NNEs} \sum_{k=1, k \neq j}^{NNEs} R(VG_i | FCR_j \text{ faulty and } FCR_k \text{ faulty}) \Pr(FCR_j \text{ faulty}) \Pr(FCR_k \text{ faulty})
 \end{aligned}
 \tag{9.21}$$

where

$$K = \begin{cases} 0, & NNEs < 5 \\ 1, & NNEs = 5 \end{cases}$$

For an AFTA consisting of multiple VGs, conditioning the VG reliabilities upon FCR fault pattern allows us to conveniently express system reliability as a summation of terms, each of which is a product of independent probabilities, or

$$\begin{aligned}
 R_{sys} &= \left[\prod_{VG_i \in F_j, F_j \in S} R(VG_i | \text{no FCR faults}) \right] \Pr(\text{no FCR faults}) \\
 &+ \sum_{n=1}^{NNEs} \left[\prod_{VG_i \in F_j, F_j \in S} R(VG_i | FCR_n \text{ faulty}) \right] \Pr(FCR_n \text{ faulty}) \\
 &+ K \sum_{n=1}^{NNEs} \sum_{m=1, m \neq n}^{NNEs} \left[\prod_{VG_i \in F_j, F_j \in S} R(VG_i | FCRs_n \text{ and } m \text{ faulty}) \right] \Pr(FCR_n \text{ faulty}) \Pr(FCR_m \text{ faulty})
 \end{aligned}
 \tag{9.22}$$

where

$$\Pr(\text{no FCR faults}) = R_{FCR}^{NNEs}$$

$$\Pr(\text{FCR } n \text{ faulty}) = \Pr(\text{FCR } m \text{ faulty}) = \binom{NNEs}{1} R_{FCR}^{NNEs-1} U_{FCR}$$

R_x = reliability of component x

$$U_x = 1 - R_x$$

$$R_{FCR} = R_{NE} R_{PC} R_{BUS}$$

$$R_{NE} = e^{-\lambda_{NE} t}$$

$$R_{PC} = e^{-\lambda_{PC} t}$$

$$R_{BUS} = e^{-\lambda_{BUS} t}$$

9.2.1. Formulation for Graceful Degradation Class of Fault Recovery

We now present a formulation for p_{GD}, the probability that all VGs needed to perform the AFTA's intended function are operational when managed according to a graceful degradation class of fault recovery policies. A given number of VGs are needed to perform the functions, and as their members fail, the VGs' redundancy levels are reduced until the VG is inoperable.

The overall analytical approach is to formulate an expression for the reliability of a VG conditioned upon a given FCR failure pattern, assuming a graceful degradation redundancy management policy. Then, the probabilities of the given FCR failure patterns are calculated.

Let $\Xi(\lambda, \mu, t, r)$ represent the reliability at mission time t of a VG having processor failure rate λ , fault recovery rate μ , and redundancy level r , assuming PE faults only. This is the probability of occurrence of all operational states (redundancy levels of 1, 2, 3, or 4) of the VG minus the probability that the VG fails due to near-coincident PE faults.

$$\Xi(\lambda, \mu, t, r) = \begin{cases} \sum_{i=1}^r \left[c_i \binom{r}{i} (e^{-\lambda t})^i (1 - e^{-\lambda t})^{r-i} \right] - \frac{r(r-1)\lambda^2 t^2}{\mu}, & r > 0 \\ 0, & r \leq 0 \end{cases} \quad (9.23)$$

where c_i is the probability that a VG of redundancy level $i+1$ can successfully degrade to a VG of redundancy level i .

If $r > 1$, then

$$c_i = \begin{cases} c_D, & i=1 \\ 1.0, & i=2 \\ 1.0, & i=3 \\ 1.0, & i=4 \end{cases}$$

If $r=1$, then

$$c_D = 1.0$$

The parameter c_D ranges from 0.5 to 0.90, depending upon the level of effort put into tolerating faults in duplex VGs. A safe assumption is usually $c_D = 0.50$, since at worst the redundancy management function can, upon detecting a fault in a duplex VG, randomly guess which one is faulty and mask it out.

Let $\text{nelist}(\text{VG}_i)$ represent the set of FCRs which contain at most one channel of VG_i . For example, if quadruply redundant VG_1 has members in FCRs 0, 1, 3, and 4, then $\text{nelist}(\text{VG}_1) = \{0, 1, 3, 4\}$.

The conditional VG reliability becomes

$$R(\text{VG}_i \mid \text{no FCR faults}) = \Xi(\lambda_{PE}, \mu_{PE}, t, \text{redlev}_i), \quad (9.24)$$

$$R(\text{VG}_i \mid \text{FCR}_j \text{ faulty}) = \begin{cases} \Xi(\lambda_{PE}, \mu_{PE}, t, \text{redlev}_i), & j \notin \text{nelist}(\text{VG}_i) \\ \Xi(\lambda_{PE}, \mu_{PE}, t, \text{redlev}_i - 1), & j \in \text{nelist}(\text{VG}_i) \end{cases}, \quad (9.25)$$

and

$$R(\text{VG}_i \mid \text{FCRs } j, k \text{ faulty}) = \begin{cases} \Xi(\lambda_{PE}, \mu_{PE}, t, \text{redlev}_i), & j \notin \text{nelist}(\text{VG}_i) \text{ and } k \notin \text{nelist}(\text{VG}_i), \\ \Xi(\lambda_{PE}, \mu_{PE}, t, \text{redlev}_i - 1), & j \notin \text{nelist}(\text{VG}_i) \text{ and } k \in \text{nelist}(\text{VG}_i), \\ \Xi(\lambda_{PE}, \mu_{PE}, t, \text{redlev}_i - 1), & j \in \text{nelist}(\text{VG}_i) \text{ and } k \notin \text{nelist}(\text{VG}_i), \\ \Xi(\lambda_{PE}, \mu_{PE}, t, \text{redlev}_i - 2), & j \in \text{nelist}(\text{VG}_i) \text{ and } k \in \text{nelist}(\text{VG}_i) \end{cases} \quad (9.26)$$

This formulation for the conditional VG reliability is used to compute PGD:

$$\begin{aligned}
 P_{GD} = & \left[\prod_{VG_i \in F, F_j \in S} R(VG_i | \text{no FCR faults}) \right] \Pr(\text{no FCR faults}) \\
 & + \sum_{n=1}^{NNEs} \left[\prod_{VG_i \in F, F_j \in S} R(VG_i | \text{FCR } n \text{ faulty}) \right] \Pr(\text{FCR } n \text{ faulty}) \\
 & + K \sum_{n=1}^{NNEs} \sum_{m=1, m \neq n}^{NNEs} \left[\prod_{VG_i \in F, F_j \in S} R(VG_i | \text{FCRs } n \text{ and } m \text{ faulty}) \right] \Pr(\text{FCR } n \text{ faulty}) \Pr(\text{FCR } m \text{ faulty})
 \end{aligned} \tag{9.27}$$

9.2.2. Formulation for Processor Replacement Class of Fault Recovery

Let NVG denote the number of VGs required to meet the mission throughput and other performance requirements, and let redlev_i denote the redundancy level required for VG_i to meet its mission reliability requirements. If NVGs of the appropriate redundancy levels cannot be formed, then AFTA cannot meet its mission requirements. In this section we produce a formulation that a desired AFTA configuration can be constructed from the PEs and NEs which are nonfaulty, given a redundancy management strategy from the processor replacement class. For the following analysis it is assumed that the term VG also includes the IOCs.

Assume that the fault-free AFTA is composed of NNE Network Elements and hence NNE FCRs. It is assumed that each VG *i* must have redundancy level redlev_i. Not all VGs need have identical redundancy levels. A total of at least

$$NPEs = \sum_{i=1}^{NVIDs} redlev_i \tag{9.28}$$

PEs are required to form NVG VGs, each VG_i of which has redundancy level redlev_i. If the AFTA is composed of 4 FCRs, then to construct the required VG configuration each FCR must contain at least

$$N_4 = \lceil NPEs/4 \rceil \tag{9.29}$$

PEs, while if the AFTA is composed of 5 FCRs, then each FCR must contain at least

$$N_5 = \lceil N_{PEs}/5 \rceil \quad (9.30)$$

PEs. Any four PEs resident in different FCRs can form a quad VG and any three a triplex VG. The probability that the requisite VGs can be formed under a processor replacement type of redundancy management strategy is

$$\begin{aligned} p_{PR} = & \Pr(\#PEs \text{ per FCR} \geq N_4) \Pr(\text{exactly 4 FCRs operational}) \\ & + \Pr(\#PEs \text{ per FCR} \geq N_5) \Pr(\text{exactly 5 FCRs operational}) \end{aligned} \quad (9.31)$$

To meet the mission requirements, each FCR must have either N_4 or N_5 PEs, depending on the number of FCRs the AFTA configuration possesses. We assume that under fault-free conditions each FCR has an equal complement of PEs. To increase availability, spare PEs may be added to each FCR to bring the total number of PEs in each FCR up to N_T . If the AFTA configuration possesses 4 FCRs, then $N_T \geq N_4$; if the configuration possesses 5 FCRs, then $N_T \geq N_5$. The probability that the requisite number of VGs can be formed is

$$\begin{aligned} p_{PR} = & \left[\sum_{n=N_4}^{N_T} \binom{N_T}{n} R_{PE}^n U_{PE}^{(N_T-n)} \right] \binom{N_{NEs}}{4} R_{FCR}^4 U_{FCR}^{(N_{NEs}-4)} \\ & + K \left[\sum_{n=N_5}^{N_T} \binom{N_T}{n} R_{PE}^n U_{PE}^{(N_T-n)} \right] R_{FCR}^5 \end{aligned} \quad (9.32)$$

where

$$R_{PE} = e^{-\lambda_{PE}t}$$

$$R_{FCR} = R_{NE} R_{PC} R_{BUS}$$

$$R_{NE} = e^{-\lambda_{NE}t}$$

$$R_{PC} = e^{-\lambda_{PC}t}$$

$$R_{BUS} = e^{-\lambda_{BUS}t}$$

$$U_x = 1 - R_x$$

and

$$K = \begin{cases} 0, & \text{NNEs} < 5 \\ 1, & \text{NNEs} = 5 \end{cases}$$

9.2.3. Failure Rate Calculation Methodology

For the purposes of reliability and availability calculations, AFTA is partitioned into LRMs and LRUs, each of which has an associated failure rate. The LRMs include processors, Network Elements, power conditioners, and input/output controllers. The LRU's primary contribution to AFTA failure rate is its backplane bus: if the bus fails, then it is assumed that the entire FCR is unusable. Secondary non-Byzantine resilient techniques may be used in a given AFTA implementation to reduce the probability of FCR backplane bus failure.

Most AFTA components are Non Developmental Items (NDI), for which failure rates and plausible calculational means should be provided by their vendor. These are usually based on MIL-HDBK-217E analyses and are furnished along with the components' documentation.

In the current analysis, we focus on the estimation and minimization of NE failure rate.

9.2.3.1. Environmental Effects

The AFTA will potentially reside in a number of different vehicles under a number of different operational and environmental conditions. These conditions must be specified for each operational mode of the system.

When possible, the MIL-HDBK-217E will be used to estimate the component failure rates of the AFTA. CECOM/RAMECES field data will also be used when available. When component failure rates are estimated using the MIL 217E handbook, the effect of the operational environment is taken into account by multiplying the component failure rate by an environmental multiplier Π_E . Values of Π_E for monolithic microelectronic devices in various operational environments are given below.

Environment	Π_E
Ground, Benign	0.38
Ground, Fixed	2.5
Manpack	3.8
Ground, Mobile	4.2
Airborne, Rotary Winged	8.5
Cannon, Launch	220.

Table 9-2. Environmental Failure Rate Multipliers for Monolithic Microelectronic Devices

9.2.3.2. PE Failure Rate Calculations

PE Mean Time Between Failures (MTBFs) are obtained from the board manufacturers and are summarized in the table below. The failure rates are assumed to be the reciprocal of the MTBFs. On the whole, the MTBFs cited by the vendors seem much higher than experience would indicate. Selected vendors were contacted for details regarding their MTBF calculation methodology, but no such information was received in time for inclusion into this report. This information would of course be required for any fielded version of AFTA as part of the vendors' documentation.

PE Type	MTBF	Environment	Π_E
Radstone PMV 68M CPU-3A	16,982h	Ground, Mobile, 45C	4.2
Lockheed Sanders STAR MVP	32,000h	Airborne, Uninhabited, 40C	6.0 [†]
SAVA GPPM	31,000h	Ground, Mobile, 85C*	4.2

Table 9-3. PE Cited Failure Rate Data

These failure rate data must be converted from the cited environment to the anticipated operational environment. The technique chosen to approximate this conversion is to mul-

[†] There are six Aircraft, Uninhabited environments specified in 217E, with Π_E s ranging from 3.0 to 9.0; the Lockheed data do not specify which is meant. The AUA, Aircraft, Uninhabited, Attack environment is assumed as it is approximately the numerical mean of all Aircraft, Uninhabited multipliers.

* This temperature is far above the 50C specified in 217E as being representative of the Ground, Mobile environment. Moreover, it is inconsistent with the SAVA maximum operating temperature specification of 78C. It is therefore assumed to be a typographic error in the draft SAVA standard.

multiply the MTBF cited by the manufacturer by the cited environment's Π_E , divided by the anticipated environment's Π_E .

9.2.3.2.1. *Hiatus: Ground Fixed*

The hiatus is assumed to occur in the Ground, Fixed environment as specified in 217E, with a Π_E of 2.5.

PE Type	MTBF	Multiplier	Π_E
Radstone PMV 68M CPU-3A	28,530h	$4.2/2.5 = 1.68$	2.5
Lockheed Sanders STAR MVP	76,800h	$6.0/2.5 = 2.40$	2.5
SAVA GPPM	52,080	$4.2/2.5 = 1.68$	2.5

Table 9-4. PE Hiatus Failure Rate Data

9.2.3.2.2. *Aircraft Mission*

The aircraft mission is assumed to occur in the Aircraft, Rotary environment as specified in 217E, with a Π_E of 8.5.

PE Type	MTBF	Multiplier	Π_E
Radstone PMV 68M CPU-3A	8,391h	$4.2/8.5 = 0.49$	8.5
Lockheed Sanders STAR MVP	22,588h	$6.0/8.5 = 0.71$	8.5
SAVA GPPM	15,190	$4.2/8.5 = 0.49$	8.5

Table 9-5. PE Aircraft Mission Failure Rate Data

9.2.3.2.3. *Ground Mission*

The ground mission is assumed to occur in the Ground, Mobile environment as specified in 217E, with a Π_E of 4.2.

PE Type	MTBF	Multiplier	Π_E
Radstone PMV 68M CPU-3A	16,982h	$4.2/4.2 = 1.0$	4.2
Lockheed Sanders STAR MVP	45,760h	$6.0/4.2 = 1.43$	4.2
SAVA GPPM	31,000	$4.2/4.2 = 1.0$	4.2

Table 9-6. PE Aircraft Mission Failure Rate Data

9.2.3.3. NE Failure Rate Calculations

9.2.3.3.1. Methodology

The AFTA Network Element failure rate is calculated using the MIL-HDBK-217E Parts Stress Analysis technique. The failure rate of two Network Element implementations will be calculated. The Baseline board is constructed of a combination of NDI ICs and the Scoreboard ASIC, as described in Section 4.5.2.2. It is believed that this is the minimum level of integration needed to allow the AFTA NE to fit on a single VMEbus-compatible or SAVA-compatible module. The High-End board consists of four ASICs, the DPRAM, and the fiber optic components, as described in Section 4.5.2.4, and represents an aggressive packaging approach which would allow the NE to readily fit on a JIAWG or smaller module. (It is likely that the NE can fit on a JIAWG module with a lower level of integration.)

9.2.3.3.2. Assumptions

The assumptions used in calculating the NE failure rate are as follows. First, the "Baseline NE," that is, the NE containing the Scoreboard ASIC, is calculated first. The High-End board is evaluated as a perturbation to the Baseline. Failure rates are calculated using the MIL-HDBK-217E Parts Stress Analysis technique. The NE is assumed to consist of Class B parts, consisting of hermetic, ceramic, eutectically bonded integrated circuits mounted to the board using plated through holes (PTH). The NE board is assumed to comply with the MIL-STD-344 form factor, and to consist of 6 signal layers. Maximum integrated circuit power dissipation specifications are used when estimating junction temperatures. We note that the maximum power dissipation is a worst-case assumption and can differ from typical power dissipation figures by factor of two. The learning factor Π_L and the voltage stress derating factor Π_V are assumed to be 1.0.

The NE failure rate comprises four main contributors: the integrated circuits, the fiber optic components or "plant", the onboard pins connecting the integrated circuits and backplane connector to the printed circuit board, and the backplane pins connecting the NE to the FCR backplane bus.

The integrated circuit failure rates are calculated according to Section 5.1.2.1 of MIL-HDBK-217E, as

$$\lambda_p = \Pi_Q (C_1 \Pi_T \Pi_V + C_2 \Pi_E) \Pi_L \text{ failures}/10^6 \text{ hours} \quad (9.33)$$

where

λ_p is the device failure rate in failures per 10^6 hours

Π_Q is the quality factor (1.0 for MIL-M-38510 Class B components)

Π_T is the temperature acceleration factor, based on technology, given by

$$\Pi_T = 0.1 \left(\exp \left[-A \left(\frac{1}{273 + (T_C + \Theta_{JC} P)} - \frac{1}{298} \right) \right] \right) \quad (9.34)$$

A = 4635 for TTL parts, 6373 for CMOS parts

T_C is the case temperature (A function of the operational environment See Table 5.1.2.7-4 in MIL-HDBK-217E.)

Θ_{JC} is the junction-to-case thermal resistance (A function of the die attach method and the number of pins. See Table 5.1.2.7-4 in MIL-HDBK-217E.)

P is the integrated circuit power dissipation

Π_V is the voltage stress derating factor (Assumed to be 1.0)

Π_E is the application environment factor

C_1 is the circuit complexity factor based on gate count and technology (See p. 5.1.2.1-1 of MIL-HDBK-217E.)

C_2 is the package complexity failure rate (See Table 5.1.2.7-16 of MIL-HDBK-217E.)

The fiber optic connector and splitter failure rates are calculated according to Section 5.1.20 of MIL-HDBK-217E. The optical emitter and detector failure rates are calculated from [APS90].

The failure rate of the onboard pins connecting the integrated circuits to the printed circuit board is calculated according to Section 5.1.13 of MIL-HDBK-217E. Note that plated through holes (PTH) are assumed to be used.

$$\lambda_p = \lambda_b \Pi_Q \Pi_E \left[n_1 \Pi_C + n_2 (\Pi_C + 13) \right] \text{ failures per } 10^6 \text{ hours}$$

where

λ_p = failure rate due to onboard pins

λ_b = base failure rate (0.000041 per 10^6 hours for wave-soldered boards, according to MIL-HDBK-217E Table 5.1.13-1)

Π_Q = quality factor (assumed to be 1.0)

Π_E = environmental factor (MIL-HDBK-217E Table 5.1.13-4)

n_1 = number of wave soldered PTHs (obtained by summing the pin counts of the integrated circuits on the NE plus the number of PTHs required to connect the backplane connector to the circuit board: 192 for VME, 256 for SAVA, and 250 for JIAWG LRMs)

n_2 = number of hand soldered PTHs (assumed to be 0)

Π_C = complexity factor (2.0 for a 6-layer NE board)

Finally, the contribution to the NE's failure rate due to the mating connector pair between the NE and the FCR backplane bus is calculated according to Section 5.1.12.2 of MIL-HDBK-217E as follows.

$$\lambda_p = \lambda_b \Pi_E \Pi_P \Pi_K \text{ failures per } 10^6 \text{ hours}$$

where

λ_b = base failure rate, given by

$$\lambda_b = 0.216 \exp \left(\frac{-2073.6}{T + 273} + \left[\frac{T + 273}{423} \right]^{4.66} \right) \text{ failures per } 10^6 \text{ hours}$$

T = operating connector temperature, C

Π_E = environmental factor (MIL-HDBK-217E Table 5.1.12.2-4)

Π_p = pin factor, given by

$$\Pi_p = \exp \left\{ \left(\frac{N-1}{10} \right)^{0.51064} \right\}$$

N = number of active pins (192 for VMEbus, 256 for SAVA, 250 for JI-AWG)

Π_K = cycling factor (cycle defined as unmating/mating of the connector, from MIL-HDBK-217E Table 5.1.12.2-7)

9.2.3.3.3. *AFTA Hiatus NE Failure Rate*

Using the calculations described above the failure rate of the baseline NE under hiatus conditions is summarized below. The first table shows the NE tentative parts list and failure rate for each part. This is a preliminary parts list which will probably change as the Detailed Design phase of the AFTA program proceeds; the AFTA analysis program will track these design refinements.

	A	H
3	CHIPS	Failures /10⁶
4	Scoreboard ASIC	1.21E+00
5	IDT 7202	1.02E+00
6	Siometrics VME controller	2.33E-01
7	Altera EPM 5064 JC-1	8.92E-01
8	Altera EPM 5128 JC-1	3.64E+00
9	Lattice GAL 18V10-15LP	3.07E-01
10	Lattice GAL 26V12-15LP	4.14E-01
11	IDT 2K x 8 SRAM IDT 6116 LA 20 TD	1.11E-01
12	IDT 4K x 8 DPRAM IDT 7134 L 35 J	2.05E-01
13	IDT 64 x 5 FIFO IDT 72402 L 25 P	2.63E-02
14	Lattice 22v10	3.84E-01
15	Altera 5032	1.12E+00
16	TI Octal Bus Transceiver	3.79E-02
17	TI Octal Trans w/Reg SN74ALS646ANT	6.99E-02
18	IDT 16k x 8 DPRAM IDT 7006 S 35 G	1.21E+00
19	IDT 2910 Microsequencer IDT 39C10C	1.01E-01
20	IDT 4k x 16 RAM w SPC IDT 71502 S 25 J	1.85E+00
21	Dallas Semiconductor watchdog timer DS1232	8.72E-03
22	AMD Taxi Transmitter AM7968-125 JC	7.18E-02
23	AMD Taxi Receiver AM7969-125 JC	2.39E-01
24	AT&T Optical Transmitter	1.63E+00
25	AT&T Optical Receiver	5.08E+00
26		1.99E+01
27		
28	PROMs	Failure Rate /10⁶
29	Cypress Registered PROMs CY7C245A-25WC	7.31E-02
30		7.31E-02
31		
32		
33	Oscillators	Failure Rate / 10⁶
34	Vectron 50 MHz oscillator CO-238A-O	8.31E-02
35		8.31E-02
36		
37	Other Parts	Failure Rate / 10⁶
38	Fiber Optic Connector	5.00E-01
39	Fiber Optic Splitter	5.00E-01
40	Optic Fiber	5.00E-01

Table 9-7. AFTA Baseline NE Parts Hiatus Failure Rates

The next table shows the total NE failure rate and how it is broken down according to the integrated circuits ("silicon" in the table), the fiber optic plant, the onboard pins, and the backplane pins. This information is depicted graphically in Figure 9-7.

4 3	Failure Rates, per 10 ⁶ hours	
4 4		
4 5	Silicon	2.00E+01
4 6	FO Plant	1.50E+00
4 7	Onboard Pins	1.90E+01
4 8	Backplane Pins	2.54E-01
4 9	Total	4.08E+01
5 0		
5 1	NE MTBF (hours)	2.45E+04

Table 9-8. AFTA Baseline NE Hiatus Failure Rate and Constituents

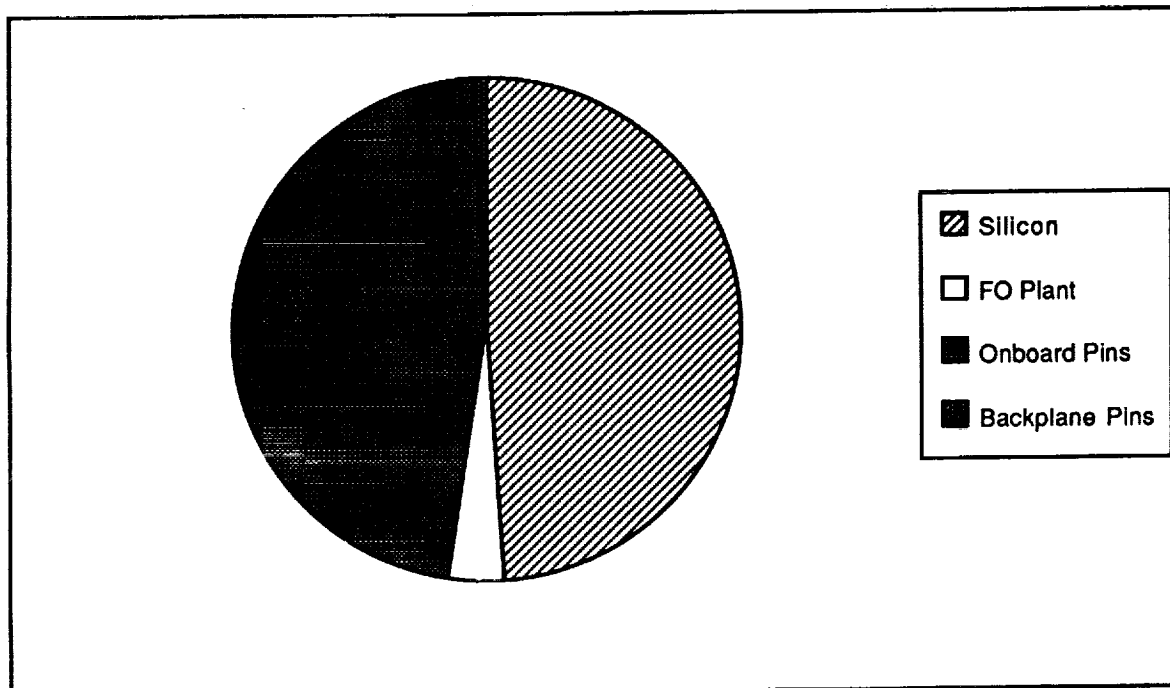


Figure 9-7. AFTA Hiatus NE Failure Rate Constituents

The calculations indicate that the Baseline AFTA network has a hiatus failure rate of 41 failures per 10⁶ hours, corresponding to an MTBF of 24,500h. From this calculation we also conclude that approximately 50% of the hiatus NE failure rate is due to onboard IC-to-board connections, and approximately 50% of hiatus NE failure rate is due to the integrated circuitry. Thus increasing the level of integration of the NE could at most increase its hiatus MTBF by a factor of two.

9.2.3.3.4. AFTA Aircraft Mission NE Failure Rate

The failure rate for the AFTA Baseline NE under Aircraft, Rotary Winged environmental conditions is presented below. For conciseness the parts list is omitted.

4 3	Failure Rates, per 10 ⁶ hours	
4 4		
4 5	Silicon	3.66E+01
4 6	FO Plant	1.50E+00
4 7	Onboard Pins	1.45E+02
4 8	Backplane Pins	1.42E+00
4 9	Total	1.85E+02
5 0		
5 1	NE MTBF (hours)	5.42E+03

Table 9-9. AFTA Baseline NE Flight Mission Failure Rate and Constituents

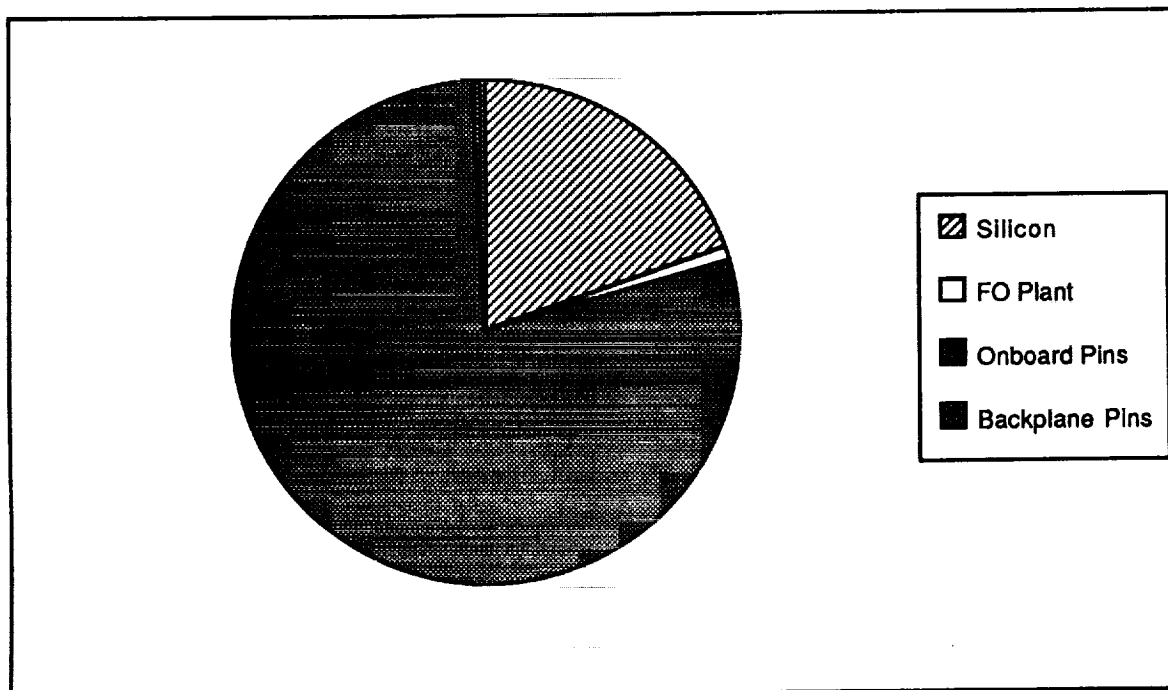


Figure 9-8. AFTA Flight Mission NE Failure Rate Constituents

In the helicopter environment, the Baseline NE failure rate has increased significantly to 185 failures per 10⁶ hours, corresponding to an MTBF of 5,400h. It is of interest to note that approximately 80% of the flight mission NE failure rate is now due to the onboard IC-to-board connections, and only 20% of the failure rate is due to the integrated circuitry. This result is most likely due to the severe vibration characteristic of the helicopter environment. The obvious implication is that increasing the level of NE integration via the use of ASICs could have a significant impact on the NE's reliability under a helicopter flight environment.

9.2.3.3.5. AFTA Ground Mission NE Failure Rate

The failure rate of the AFTA Baseline NE under Ground, Mobile conditions is presented below.

	A	B
4 3	Failure Rates, per 10 ⁶ hours	
4 4		
4 5	Silicon	2.43E+01
4 6	FO Plant	1.50E+00
4 7	Onboard Pins	6.62E+01
4 8	Backplane Pins	6.19E-01
4 9	Total	9.26E+01
5 0		
5 1	NE MTBF (hours)	1.08E+04

Table 9-10. AFTA Baseline NE Ground Mission Failure Rate and Constituents

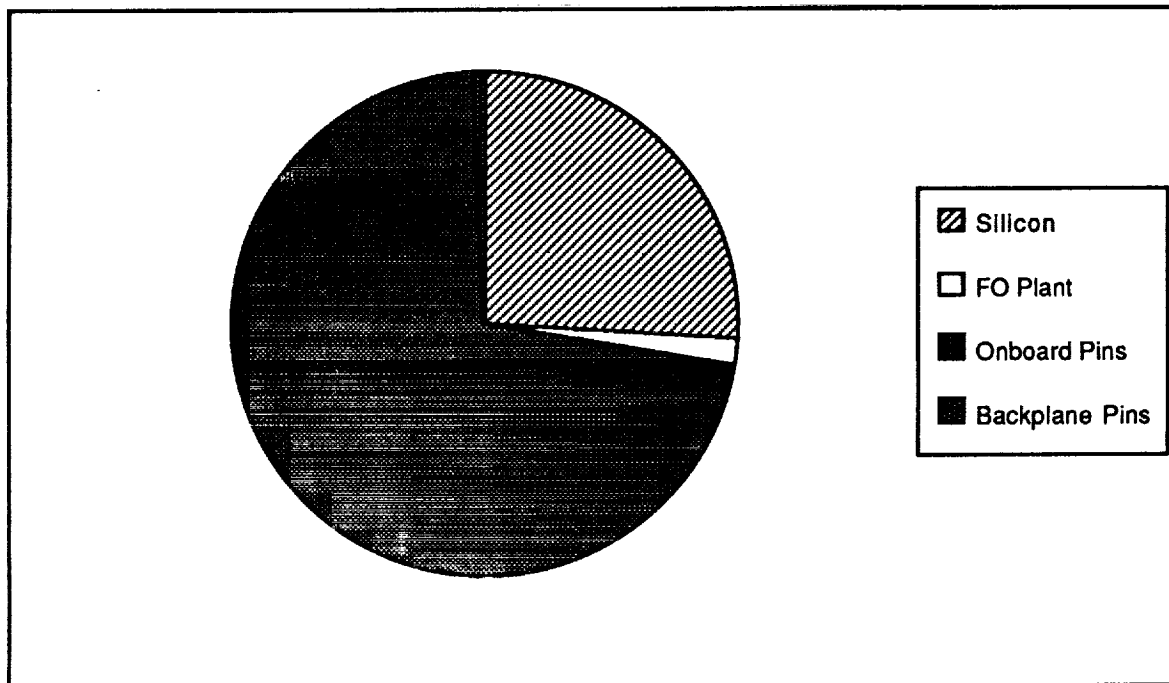


Figure 9-9. AFTA Ground Mission NE Failure Rate Constituents

The AFTA Baseline Ground Mission NE failure rate is estimated to be 93 failures per 10⁶ hours, corresponding to an MTBF of 10,800h. Of this failure rate, approximately 72% is due to onboard pins, with 26% due to the integrated circuitry. Again, a significant benefit can be obtained through from reducing the number of onboard pins through increasing the level of integration of the silicon.

9.2.3.3.6. *Implications and Indicated Course of Action*

The AFTA Baseline failure rate calculations are summarized below.

Environment	MTBF, h	% due to ICs	% due to onboard pins
Ground, Fixed	24,500	50	50
Aircraft, Rotary Winged	5,400	20	80
Ground, Mobile	10,800	26	72

Table 9-11. Summary of AFTA Baseline NE MTBF Data

These results imply that a higher level of circuitry integration could reduce the hiatus NE failure rate by up to 50% and the mission NE failure rate by at most 80%, assuming that an increased level of integration does not reduce the reliability of the silicon.

This motivates repetition of the calculation using a partitioning of the NE into VH-SIC/VLSI ASICs corresponding to the partitioning described as the “High End Network Element” in Section 4. In this partitioning, the NE is comprised of four ASICs and 16K x 32 bits of DPRAM. The four ASICs correspond to the Scoreboard, the Global Controller, the Voter/FTC, and the VME Controller. The gate count, pin count, and power dissipation estimates are presented in the table below. For a detailed breakdown of the constituents of these devices, refer to Section 4.5.2.5. Also note that the largest devices, notably the Global Controller and Dual-Ported RAM (DPRAM), are mostly RAM and ROM.

Device	# Gates	# Pins	Power Consumption, W
Scoreboard	130K	145	2.172
Global Controller	658K	80	0.675
Voter/FTC	55K	193	1.370
VME Controller	156K	131	1.026
DPRAM	1M	272	2.5
Total	2M	821	7.8

Table 9-12. Gates, Pins, and Power Consumption of High-End NE

MIL-HDBK-217E formulations do not extend to gate counts of highly integrated (e.g., 30,000+ gate) ASICs, so other analytical means will be required to obtain plausible failure rates. Specifically, the devices will be broken up into logic gate count, RAM bit count,

and package complexity, the failure rate for each segment of the ASIC will be computed separately, and the three will be combined. Failure rate due to logic gates will use the formulation

$$\lambda_{p, \text{logic}} = \Pi_Q (C_{1, \text{logic}} \Pi_T \Pi_V) \Pi_L \text{ failures}/10^6 \text{ hours}$$

failure rate due to RAM will use the formulation

$$\lambda_{p, \text{RAM}} = \Pi_Q (C_{1, \text{RAM}} \Pi_T \Pi_V) \Pi_L \text{ failures}/10^6 \text{ hours}$$

and failure rate due to package complexity will use the formulation

$$\lambda_{p, \text{package}} = \Pi_Q C_2 \Pi_E \Pi_L \text{ failures}/10^6 \text{ hours}$$

where, Π_Q , Π_T , Π_V , Π_E , Π_L , and the failure rate formulations due to onboard pins, fiber optic components, and backplane connectors are unchanged from previous calculations. Because of the large number of pins on each package, Θ_{JC} is estimated as 25C/W according to MIL-HDBK-217E.

Scoreboard

50K logic gates	$C_{1, \text{logic}} = 0.24$ (extrapolated from MIL-HDBK-217E)
40Kbits RAM	$C_{1, \text{RAM}} = 0.20$
package	$C_2 = 0.076$

The device failure rate is, in failures per 10^6 hours

GF	AR	GM
3.4	6.9	4.4

where, as usual, GF refers to the Ground, Fixed hiatus environment, AR refers to an Aircraft, Rotary Wing mission environment, and GM refers to a Ground, Mobile environment. Note that this Scoreboard ASIC has a higher failure rate than that in the Baseline NE. This is because the Baseline Scoreboard ASIC does not have RAM integrated on it as does the High End Scoreboard ASIC.

Global Controller

2.5K logic gates $C_{1,logic} = 0.04$

328Kbits RAM $C_{1,RAM} = 0.60$ (extrapolated from MIL-HDBK-217E)

package $C_2 = 0.032$

The device failure rate is, in failures per 10^6 hours

GF	AR	GM
0.8	1.9	1.1

Voter/FTC

30K logic gates $C_{1,logic} = 0.16$

12Kbits RAM $C_{1,RAM} = 0.10$

package $C_2 = 0.076$

The device failure rate is, in failures per 10^6 hours

GF	AR	GM
0.9	2.2	1.3

VME Controller

24K logic gates $C_{1,logic} = 0.16$

66Kbits RAM $C_{1,RAM} = 0.40$

package $C_2 = 0.053$

The device failure rate is, in failures per 10^6 hours

GF	AR	GM
1.1	2.5	1.4

DPRAM

512Kbits RAM $C_{1,RAM} = 0.80$

package

$$C_2 = 0.10$$

The device failure rate is, in failures per 10^6 hours

GF	AR	GM
8.5	16.4	10.7

The total NE Device failure rates are, in failures per 10^6 hours,

GF	AR	GM
14.7	29.9	18.9

The single largest contributor of High End ASIC failure rate is the DPRAM (58%, 55%, and 57% for the three mission environments). The total number of onboard pins in the High-End NE is 1021 (821 connecting the ASICs to the board and approximately 200 connecting the backplane connector to the board), compared to 1682 for the Baseline NE. The NE's power dissipation is also reduced from 40W to 13.4W.

The net result of the integration of the NE circuitry into VHSIC/VLSI ASICs is as follows.

The hiatus failure rate for the High-End board is

Component	Failure Rate, per 10^6 hours
Silicon	14.7
FO Plant	1.5
Onboard Pins	10.5
Backplane Pins	0.25
Total Failure Rate	27
MTBF	37,106h

The Aircraft Mission failure rate for the High-End board is

Component	Failure Rate, per 10 ⁶ hours
Silicon	29.9
FO Plant	1.5
Onboard Pins	80
Backplane Pins	1.42
Total Failure Rate	113
MTBF	8,863h

The Ground Vehicle Mission failure rate for the High-End board is

Component	Failure Rate, per 10 ⁶ hours
Silicon	18.9
FO Plant	1.5
Onboard Pins	36.4
Backplane Pins	0.6
Total Failure Rate	57.4
MTBF	17,421h

The following table compares the MTBFs of the AFTA Baseline NE MTBF and the AFTA High End NE, for the three operational environments under consideration. The improvement factor is calculated as the ratio of the High End NE MTBF to the Baseline NE MTBF.

Environment	Baseline NE	High End NE	Improvement Factor
Ground, Fixed	24,500h	37,106h	1.51
Aircraft, Rotary Winged	5,400h	8,863h	1.64
Ground, Mobile	10,800h	17,421h	1.61

Table 9-13. Comparison of MTBFs of Baseline and High End AFTA Network Element

9.2.3.4. IOC Failure Rate Calculations

As for all NDI AFTA modules, IOC failure rates must be provided by the manufacturer of the modules or calculated from detailed design information as outlined above. Currently, no IOCs have been definitively selected for inclusion in AFTA and therefore no failure rate data are available. As the Detailed Design phase proceeds and the AFTA I/O suite

is definitized, these data will be obtained and incorporated into the AFTA analytical model suite.

9.2.3.5. PC Failure Rate Calculations

Power conditioner failure rates for militarized VMEbus- and SAVA-compatible modules were not obtained under the Conceptual Study phase of the AFTA program. However, [MA-HDBK] contains a list of JIAWG-like power conditioners and failure rate data. We make the approximation that a PC packaged in VMEbus or SAVA form factor will have a similar failure rate, and use the following PC failure rate data.

PC Type	MTBF	Environment	Π_E
Varo Power Systems Model 24039	20,000h	Airborne, Uninhabited, Fighter, 71C	9.0
General Dynamics Model PS32	15,850h	Airborne, Uninhabited, Fighter*	9.0

Table 9-14. PC Cited Failure Rate Data

For the AFTA Conceptual Study analysis we assume the existence of a generic PC having an MTBF of 17,500h under the Airborne, Uninhabited, Fighter environment. As usual, a module's failure rate is assumed to be the reciprocal of its MTBF.

9.2.3.5.1. Hiatus

The hiatus MBTF of the generic AFTA PC is estimated as

$$MTBF_{PC,hiatus} = 17,500 \left(\frac{9.0}{2.5} \right) = 63,000h$$

9.2.3.5.2. Aircraft Mission

The aircraft mission MBTF of the generic AFTA PC is estimated as

$$MTBF_{PC,aircraft} = 17,500 \left(\frac{9.0}{8.5} \right) = 18,529h$$

* Environment uncited: assumed to reside in F-16.

9.2.3.5.3. Ground Mission

The ground mission MBTF of the generic AFTA PC is estimated as

$$MTBF_{PC,ground} = 17,500 \left(\frac{9.0}{4.2} \right) = 37,500h$$

9.3. Physical Characteristics (WPV) Models

The weight, power, and volume (WPV) models are simple linear summations of the WPV of the LRMs and LRUs comprising a given AFTA configuration.

9.3.1. Weight

The weight of AFTA is given by

$$WT_{AFTA} = \sum_i WT_{FCRi} \quad (9.35)$$

where WT_{FCRi} , the weight of FCR i, is given by

$$WT_{FCRi} = N_{PEi}WT_{PE} + N_{NEi}WT_{NE} + N_{IOCi}WT_{IOC} + WT_{TRACK} + N_{PCi}WT_{PC} \quad (9.36)$$

9.3.2. Power

The power consumption of AFTA is given by

$$PWR_{AFTA} = \sum_i PWR_{FCRi} \quad (9.37)$$

where PWR_{FCRi} , the power consumption of FCR i, is given by

$$PWR_{FCRi} = N_{PEi}PWR_{PE} + N_{NEi}PWR_{NE} + N_{IOCi}PWR_{IOC} + PWR_{BT} + PWR_{PC} \quad (9.38)$$

$$PWR_{PC} = (1 - \epsilon_{PC}) * (N_{PEi}PWR_{PE} + N_{NEi}PWR_{NE} + N_{IOCi}PWR_{IOC}) \quad (9.39)$$

and

$$\epsilon_{PC} = \text{Power Conditioner Efficiency}$$

Since we are assuming that no power-down standby redundancy is being used in the AFTA, the peak power and the average power are identical.

9.3.3. Volume

The volume (or size) of AFTA is given by

$$VOL_{AFTA} = \sum_i VOL_{FCRi} \quad (9.40)$$

where VOL_{FCRi} , the volume of FCR i, is given by

$$VOL_{FCRi} = N_{PEi}VOL_{PE} + N_{NEi}VOL_{NE} + N_{IOCi}VOL_{IOC} + VOL_{RACK} + N_{PCi}VOL_{PC} \quad (9.41)$$

In a modular system such as AFTA it is usually most convenient to specify the volume of a module in terms of the number of "slots" it occupies, accompanied by the volume of a slot.

9.4. Fleet Life-Cycle Cost per Service Unit (FLCCPSU) Model

It is desirable at the Conceptual Study stage to conduct a preliminary quantification of the effect of varying AFTA parameters such as VG redundancy level, sparing, redundancy management policy, implementation technology, etc. on operational and logistics costs. Military missions are complex and a full cost model must reflect this complexity; however, at the Conceptual Study phase little information is available about mission details, while in turn a full mission life cycle cost analysis is beyond the scope of a Conceptual Study. Consequently, a simple Fleet Life-Cycle Cost per Service Unit (FLCCPSU) model will be used to illustrate the effects of varying AFTA parameters on overall cost. The FLCCPSU model computes the cost of vehicle and AFTA procurement, the cost due to repairing and/or replacing spare components, and the cost due to mission-critical failures.

$$\begin{aligned} \text{FLCCPSU} = & \text{Initial procurement cost} + \\ & \text{Cost due to repair actions} + \\ & \text{Cost due to replacement parts} + \\ & \text{Cost due to unreliability} \end{aligned} \quad (9.42)$$

The constituents of these costs are described below.

9.4.1. Assumptions and Analysis Inputs

The FLCCPSU model presented below is primarily relevant to an iterative aircraft mission, in which a fleet of vehicles must periodically sortie, perform a mission, and return to base. This scenario is described in Section 2, and a figure from that section illustrating the mission scenario is reproduced below.

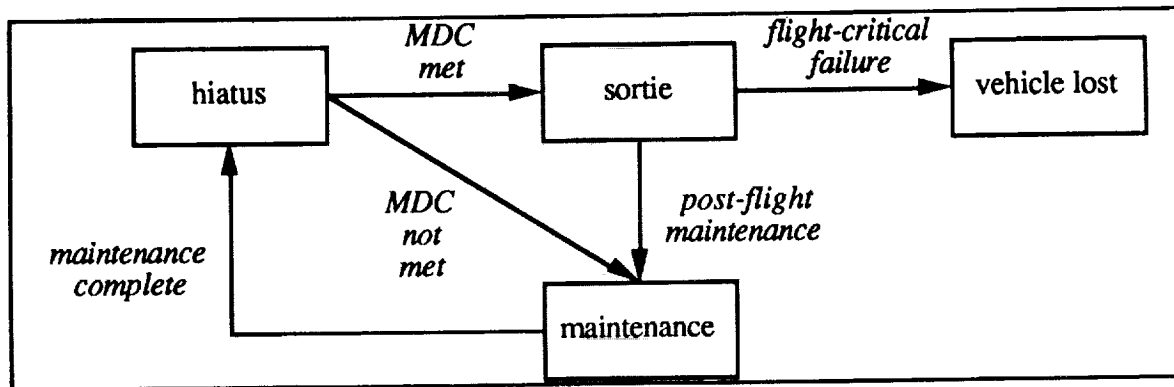


Figure 9-10. Helicopter TF/TA/NOE Mission Scenario State Diagram

Failure on the part of AFTA to form MDC can prevent a vehicle to sortie; if it is assumed that a given number of vehicles must sortie, then additional vehicles must be procured to achieve this given level of readiness. This is the cost due to unavailability, and is included in the formulation for initial procurement cost. Failure on the part of AFTA to perform flight-critical functions during the mission results in loss of the vehicle; procurement of lost vehicles (not to mention crews) contributes to the cost of maintaining the given readiness level. This is the cost due to unreliability. Faults occurring either between or during missions require maintenance actions. The manpower involved in performing the maintenance actions and the cost involved in refurbishing or reprocurring failed AFTA modules contributes to the life cycle cost. of the fleet.

The following list enumerates the input parameters to the FLCCPSU model.

Fleet Service Life, T_s : It is assumed that the fleet is in continuous operation over the Fleet Service Life.

Hiatus duration = T_h : The vehicle is in a stand down mode, i.e., powered off and unoccupied, during the hiatus period of T_h hours.

Sortie duration = T_m : The sortie lasts for T_m hours.

Number of vehicles required per sortie = N_{vs} : N_{vs} vehicles must sortie for each mission.

Baseline vehicle cost = C_{vehb} : A vehicle (without AFTA) costs C_{vehb} dollars.

Cost of an AFTA LRM = C_{module} : An AFTA module costs C_{module} dollars. Currently, it is assumed that all AFTA modules cost the same.

Number of LRMs in AFTA = N_{LRM}

Cost of an AFTA rack = C_{rack} : An empty AFTA LRU costs C_{rack} dollars.

Number of racks in AFTA = N_{rack}

Cost to repair a faulty LRM found after hiatus = $C_{repair,hiatus}$: The manpower cost to repair a faulty LRM found after hiatus is $C_{repair,hiatus}$ dollars.

Cost to repair a faulty LRM found after sortie = $C_{repair,sortie}$: The manpower cost to repair a faulty LRM found after hiatus is $C_{repair,sortie}$ dollars.

Field repair time = t_m : The time required to perform field maintenance and repair of a faulty AFTA component is t_m hours.

Field diagnosis time = $t_{diag, field, nf}$: The time required to perform field diagnosis of a AFTA component found to be not faulty is $t_{diag, field, nf}$ hours.

Field repair man hour cost = C_{mh} : The cost per man hour to perform field diagnosis, maintenance, and repair of a faulty AFTA component is C_{mh} dollars.

Depot diagnosis time = $t_{diag,depot}$: The time required to perform depot-level diagnosis of a faulty AFTA component is $t_{diag,depot}$ hours.

Depot diagnosis man hour cost = $C_{diag,depot}$: The cost per man hour to perform depot-level diagnosis of a faulty AFTA component is $C_{diag,depot}$ dollars.

Depot repair time = $t_{repair,depot}$: The time required to perform depot-level repair of a faulty AFTA component is $t_{repair,depot}$ hours.

Depot repair man hour cost = $C_{\text{repair,depot}}$: The cost per man hour to perform depot-level repair of a faulty AFTA component is $C_{\text{repair,depot}}$ dollars.

Depot condemnation ratio for an AFTA LRM = r_{condemn} : The conditional probability that the depot can not refurbish a returned module is r_{condemn} .

Cost of refurbishing an AFTA LRM = $C_{\text{refurbish}}$: The cost of the parts necessary to repair an AFTA LRM is $C_{\text{refurbish}}$ dollars. Currently, it is assumed that all AFTA modules cost the same to refurbish.

9.4.2. Application Scenario

It is assumed that the hiatus phase begins with no faults in AFTA. After T_h hours, all vehicles in the fleet attempt to sortie. Vehicles able to sortie because they can form MDC perform a sortie of T_m hours, while vehicles failing to form MDC do not sortie. Vehicles suffering faults either during the hiatus or sortie are repaired before the hiatus period begins again. This cycle is repeated for the entire service life of the fleet (or until the maintenance and flight crews mutiny).

N_{fs} , the total number of sorties required per fleet over the fleet's service life, is equal to N_{vs} , the number of vehicles per sortie, times $T_s/(T_h + T_m)$, the number of sorties per vehicle over the fleet's service life.

$$N_{fs} = N_{vs}(T_s/(T_h + T_m)) \quad (9.43)$$

9.4.3. Procurement Cost

The total number of vehicles procured is the number of vehicles required to meet the sortie requirement, N_{vs} , divided by the availability of AFTA. The availability of AFTA is given by $ppr(T_h)$, the probability that MDC can be formed under the processor replacement class of redundancy management strategies described in Section 5:

$$N_{vp} = N_{vs}/ppr(T_h) \quad (9.44)$$

The baseline cost of the vehicle is C_{vehb} , and the cost of AFTA is C_{AFTA} , where

$$C_{AFTA} = N_{LRM}C_{\text{module}} + N_{\text{rack}}C_{\text{rack}} \quad (9.45)$$

The total cost of the vehicle is

$$C_{veh} = C_{AFTA} + C_{vehb} \quad (9.46)$$

The total cost of vehicles procured to meet the sortie requirement is

$$C_{vp} = N_{vp} C_{veh} \quad (9.47)$$

9.4.4. Manpower Cost due to Repairs

The expected number of faulty AFTA LRMs in a single vehicle after a single hiatus, assuming that all LRMs have approximately the same failure rate and racks do not fail, is

$$f_h = \sum_{j=1}^{N_{LRM}} j(1-r_m(T_h))^j r_m(T_h)^{(N_{LRM}-j)}, r_m(t) = e^{-\lambda_m t} \quad (9.48)$$

The expected number of faulty AFTA LRMs in a single vehicle after a single sortie is

$$f_m = \sum_{j=1}^{N_{LRM}} j(1-r_m(T_m))^j r_m(T_m)^{(N_{LRM}-j)}, r_m(t) = e^{-\lambda_m t} \quad (9.49)$$

The expected cost of field repairs (not counting parts) for a single vehicle for a single hiatus/sortie interval

$$C_{repair,field,ss} = f_h C_{repair,hiatus} + f_m C_{repair,sortie} \quad (9.50)$$

where "ss" stands for "single sortie." We make the reasonable assumption that $C_{repair,hiatus} = C_{repair,sortie}$. To perform field repair, maintenance crew time must be spent testing, identifying, and replacing the faulty module according to the procedure described in Section 6. Denote this maintenance and repair time by t_m . With a field maintenance crew manpower cost per hour of C_{mh} , the field maintenance manpower cost per vehicle per sortie is

$$C_{repair,field,ss} = (f_h + f_m) t_m C_{mh} \quad (9.51)$$

The expected manpower cost of field repairs for the entire fleet over its entire service life

$$C_{repair,field,fleet} = N_{fs} C_{repair,field,ss} \quad (9.52)$$

After a defective module has been removed from the vehicle by field maintenance personnel, to maintain a given level of readiness the spare module used for replacement must itself be replaced. To achieve this objective two options exist. The defective module may be condemned and discarded, in which case a spare module must be procured. Alternatively, the defective module may be shipped back to a depot for repair. We assume that modules are condemned and discarded either at the field or the depot with probability r_{condemn} and successfully repaired with probability $1 - r_{\text{condemn}}$. If the module is condemned in the field, the manpower cost in doing so is negligible; other than the inevitable paperwork the field crew is assumed to throw the defective module in the trash. If the module is shipped back to the depot, shipping costs are incurred, and the depot maintenance technician must perform additional testing to determine whether the module is repairable using depot level tests. If we neglect shipping costs, we can without numerical error assume that the field crew always ships the module back to the depot for further diagnosis. We assume that the depot diagnostics require $t_{\text{diag,depot}}$ hours at a cost of $C_{\text{diag,depot}}$ per diagnostic hour. If the module is condemned as a result of these tests, negligible additional cost is incurred other than paperwork time. If the module is repaired, additional manpower and parts costs are incurred. After diagnosis, repair requires $t_{\text{repair,depot}}$ time, at a cost of $C_{\text{repair,depot}}$ per hour. In addition, spare parts (such as integrated circuits) are required to replace those found to be faulty on the module. This cost will be accounted for in the subsequent Section. Putting this all together, the total manpower cost per vehicle per sortie for depot repair of a faulty module is

$$C_{\text{repair,depot,ss}} = (f_h + f_m)(t_{\text{diag,depot}}C_{\text{diag,depot}} + (1 - r_{\text{condemn}})t_{\text{repair,depot}}C_{\text{repair,depot}}) \quad (9.53)$$

The expected manpower cost of depot repairs for the entire fleet over its entire service life is

$$C_{\text{repair,depot,fleet}} = N_{fs}C_{\text{repair,depot,ss}} \quad (9.54)$$

9.4.5. Cost due to Spares

The expected number of LRMs which must be replaced for the entire fleet over its service life is equal to the expected number of faulty modules per sortie, times the number of fleet sorties. Denoting this quantity by $N_{\text{spares,fleet}}$,

$$N_{\text{spares,fleet}} = N_{fs}(f_h + f_m) \quad (9.55)$$

The cost of spare parts required to maintain the AFTA fleet over its service life is the number of module faults incurred during the fleet service life times the cost of replacing a module or repairing it at the depot.

$$C_{\text{spares, fleet}} = N_{\text{spares, fleet}}(r_{\text{condemn}}C_{\text{module}} + (1-r_{\text{condemn}})C_{\text{refurbish}}) \quad (9.56)$$

9.4.6. Cost due to Unreliability of AFTA

Assuming that the crew escapes the effects of loss of vehicle-critical computing functions, the cost due to unreliability of the AFTA is the total number of fleet sorties over the service life times the probability that the AFTA causes the loss of a vehicle during a single sortie times the cost of the vehicle.

$$C_{\text{ur}} = N_{\text{fs}}(1-p_{\text{GD}}(T_m))C_{\text{veh}} \quad (9.57)$$

where p_{GD} is the probability that the flight-critical computing functions can be performed by AFTA when its redundant components are managed according to the graceful degradation class of redundancy management policies.

9.4.7. Total FLCCPSU

The Fleet Life Cycle Cost per Service Unit is

$$\text{FLCCPSU} = C_{\text{vp}} + C_{\text{repair, field, fleet}} + C_{\text{repair, depot, fleet}} + C_{\text{spares, fleet}} + C_{\text{ur}} \quad (9.58)$$

10. VHSIC Hardware Description Language

The VHSIC Hardware Description Language (VHDL) is rapidly becoming a standard tool for digital logic design. Using VHDL, an engineer can write initial design specifications, device behavioral characteristics, device structural characteristics, and device timing characteristics using a single, integrated design environment. This section details the usefulness of VHDL for the AFTA brassboard development project, including the areas of logic design and synthesis, simulation, testing, and documentation for reprourement.

10.1. VHDL Overview

VHDL (VHSIC Hardware Description Language) is a hardware design language developed under the VHSIC (Very High-Speed Integrated Circuit) project. The language can be used for a number of applications, including design, debugging, simulation, testing, performance analysis, and documentation. The use of VHDL is required by the Department of Defense for all new ASIC designs built to military specifications. The specification of the VHDL language is standardized in [IEEE1076].

Traditional idealized design procedures prescribe a top-down design methodology. The goal is to define a design at an abstract level, then traceably refine each major component into increasing levels of complexity until a description of the design suitable for fabrication is produced. In practice, this ideal is rarely observed. Instead, the designer may pursue a bottom-up, middle-out, or a combination of the three approaches. VHDL supports the designer in any of these approaches.

Designs described by VHDL can be simulated before the design is constructed. The simulation environment is a defined part of the language. VHDL is not a static language, which just defines interconnections between elements. Processes, which can contain logic, state, and timing relations can also be defined. The simulator uses these constructs to "run" the design. A VHDL simulator can provide a viewport into a VHDL model, possibly through a source-level debugger.

The VHDL resulting from the design process can be used as a form of self-documentation. The highly abstract description (called the behavioral model) defines the functions of the design. The low-level description (called the structural model) defines the interconnect between individual hardware pieces.

10.1.1. Behavioral vs. Structural Models

VHDL supports the varying of abstraction by the designer during the design process. The mechanism for varying the design abstraction is done by using two types of models: behavioral and structural. A particular VHDL description can not be truly classified as either behavioral or structural. Part of the power of VHDL is the ability to mix structural and behavioral models. In fact, all VHDL descriptions include some behavioral modeling.

A VHDL description can be represented as a tree structure as shown in Figure 10-1. A structural VHDL model contains interconnected instantiations of other VHDL models. This is analogous to describing a circuit as a netlist, with device pins (ports in VHDL) connected to wires (signals in VHDL). A behavioral VHDL model describes the operation of a particular device in terms of state, output and timing relations as a function of device inputs. The lowest level, or leaf-level, models in any VHDL description are always behavioral models.

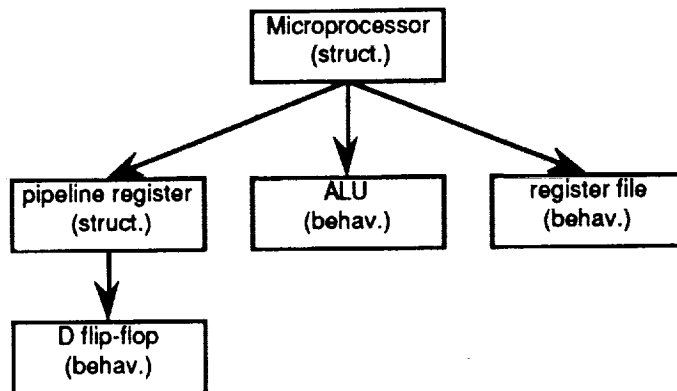


Figure 10-1. Hierarchical VHDL Model

An example of a behavioral model for a D flip-flop is shown in Figure 10-2. The behavioral model of the flip-flop defines the state of the outputs Q and !Q as a function of the clock and D inputs. The behavioral model can also include the timing relations between the clock and the outputs (clock to Q propagation delay) and the timing relations between the D input and the outputs (setup and hold times). A properly developed behavioral model would check for violations of setup time, hold time, pulse width minimums, etc., with any violations reported to the user. The development of accurate behavioral models with these characteristics is a very tedious, labor-intensive, time-consuming, and expensive task.

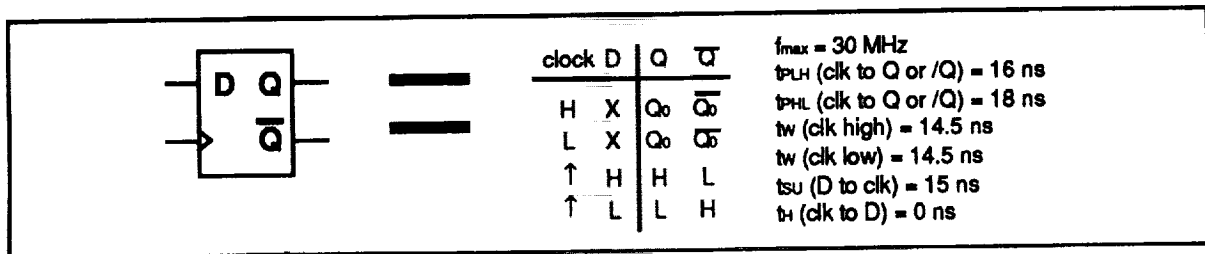


Figure 10-2. Behavioral Model of a D Flip-Flop*

The flip-flop can also be represented as a structural model defining instantiations of logic gates and interconnections between the logic gates as shown in Figure 10-3. In this case, the logic gates would be represented by behavioral models defining the device function (truth-table) and timing (propagation delay) as shown in Figure 10-4.

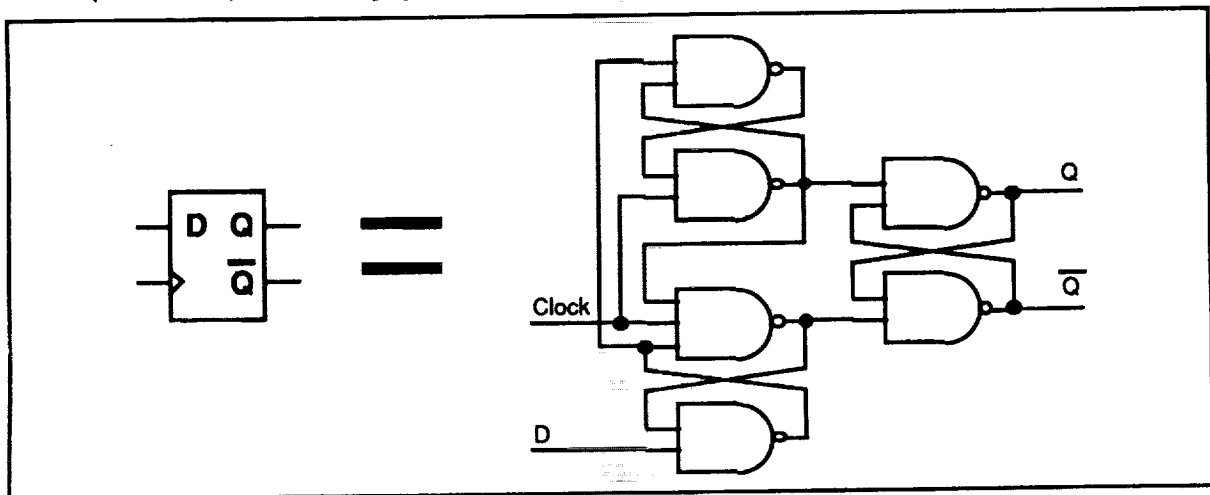


Figure 10-3. Structural Model of a D Flip-Flop†

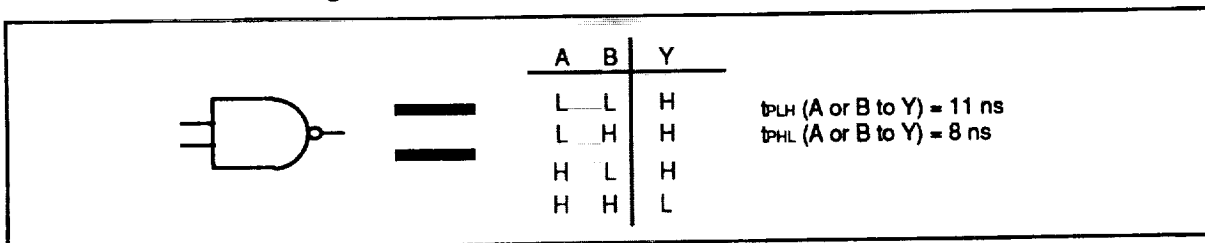


Figure 10-4. Behavioral Model of a NAND Gate

The behavioral description of the flip-flop is sufficient in most instances to define the function of the circuit. The structural model is too low level for most applications. The

* From Texas Instruments ALS/AS Logic Data Book 1986.

† From Texas Instruments TTL Logic Standard TTL, Schottky, Low-Power Schottky Data Book 1988.

function of the flip-flop is readily apparent from the behavioral description, but not from the structural description. The flip-flop behavioral model could be instantiated in higher level structural models. The flip-flop structural model, however, would probably only be used by an engineer to design the flip-flop itself.

10.1.2. Overview of a VHDL Description

A VHDL model is composed of a number of different elements, including packages, entities, architectures, and configurations. Each element is contained in a library. An element can access other elements by specifying the library name containing the referenced element and the element name. One library, known as "WORK", is used to store models under development. Other libraries may be used by explicitly declaring their usage within a model.

A package contains certain constants, type declarations, global signals, functions, and procedures that are used throughout a model. Examples of packages include TEXTIO, and STANDARD; these packages are defined as part of the VHDL standard environment [IEEE1076]. Packages are convenient for hiding definitions from the designer and for allowing incremental changes. A package declaration can be kept separate from the package body. The package declaration (referred to in VHDL simply as the package) contains declarations for the contents of the package body. The package body actually contains the definitions. Changes to, and recompilation of the package body does not require recompilation of models which use the package.

An entity describes the interface to the model. The entity is analogous to a symbol, such as the flip-flop symbol shown in Figure 10-2. Ports for input and output signals are defined in the entity. The entity also allows the definition of generics, which provide a method of passing parameters into the model. Different instantiations of a model can have different signals attached to the ports and different values assigned to the generics.

The architecture defines the model, either in behavioral or structural terms. A behavior model, such as that shown in Figure 10-2, can be represented as a set of one or more concurrent processes. A structural model can be represented as a schematic, such as that shown in Figure 10-3 for the D flip-flop. A structural architecture defines instantiations of other entities, which can themselves be either structural, behavioral, or a combination of the two. Note that a model can have more than one architecture defining the model. One architecture is selected to represent the model in a particular situation.

The selection of architectures is done using a configuration. The configuration tells the VHDL compiler/analyzer which architecture to "plug into" each entity instantiation. Generics can also be defined in the configuration section. The configuration can define the entire VHDL description hierarchically, or it can call other configurations to define lower level structural models.

10.2. Use of VHDL for AFTA

VHDL will be used extensively in the AFTA Network Element (NE) design. A VHDL analysis tool will be used to design, analyze, simulate, and document the Network Element. VHDL descriptions of the Network Element subsections can also be used as a basis for performing validation of the Network Element design. VHDL will also be used as the medium for converting appropriate subsections into application-specific integrated circuits (ASICs).

VHDL analysis and simulation is rapidly becoming a standard tool for performing chip, board, and system design. The flexibility of VHDL permits almost any digital circuit to be described and simulated using standard VHDL analysis tools. This capability permits debugging of circuit designs before building them. If accurate models are used in the analysis, the resulting simulations will be highly faithful to the actual observed behavior of the final design.

Subsections of the AFTA NE design which may require a significant amount of design and testing include the scoreboard and the fault-tolerant clock. Other sections which will benefit from the use of VHDL are the data path voter, the global controller, and the ring buffer manager.

The VHDL description of the scoreboard is particularly important since the scoreboard is a prime candidate for implementation in an ASIC device. Since an ASIC represents a significant investment and is not easily changed, the scoreboard design must be robust. The VHDL analysis tool will permit a significant amount of testing to verify the correct functionality of the scoreboard design. The scoreboard register-transfer level (RTL) VHDL description can be converted, with the use of a synthesis tool, directly to an ASIC design. The test bench created to test the scoreboard description can also be used to generate functional test vectors for testing the scoreboard ASIC during foundry testing.

10.2.1. Design

One of the primary uses of VHDL for the AFTA Network Element is for conceptual and detailed design of custom hardware. VHDL is a useful tool for performing detailed design, supplementing conventional tools such as schematic capture and Boolean equation synthesis tools. Conceptual design at a very high level can also be done in VHDL, something for which no comparable tool exists. The transformation from high level to detailed design can be performed completely within the VHDL environment.

We shall use the following definitions within the scope of the AFTA NE development.

Definition 1:

Behavioral VHDL is defined to be a VHDL architecture which uses any of the legal VHDL constructs, including those which do not correspond to possible hardware realizations of the description (i.e., pure behavioral may not be synthesizable).

Definition 2:

Structural VHDL is defined to be a VHDL architecture that consists strictly of instantiations of other entities and the interconnect between these entities.

Definition 3:

Register Transfer Level (RTL) VHDL is defined to be synthesizable behavioral VHDL, that is, a behavioral VHDL description that is suitable for input to a synthesis tool.

10.2.1.1. Behavioral

The architecture of submodules in the Network Element will be developed using VHDL behavioral modeling. Alternate partitioning of the designs can be done using behavioral models. Simulations using alternate partitioning can be used to determine what partitioning provides optimal performance.

The behavioral models will be decomposed into register-transfer level (RTL) descriptions. Some models will be developed only in RTL form. The RTL form is also a behavioral format which specifies the functionality of a block from the standpoint of random combinational logic and/or synchronous registers. Synthesis of a gate-level design from an RTL description is a straightforward process using (expensive) logic synthesis tools.

10.2.1.2. Structural

The structural description of Network Element submodules specifies the design of the submodule as an interconnection of lower level components. The gate-level netlists synthesized from RTL behavioral descriptions represent structural descriptions.

Structural VHDL requires accurate models defining the behavior of the individual components. For example, an ASIC design would require high fidelity models defining each gate, register, and macrocell used by the design. A design utilizing standard devices also requires models defining the behavior of PALs, PROMs, and other MSI/LSI parts. The development of structural descriptions of the AFTA Network Element is contingent upon the availability of these models. Structural descriptions of subsections for which suitable models are not available would not be a useful exercise, since the description would not represent any characteristics not already described more concisely by the behavioral description.

10.2.2. Simulation

One of the most useful features of VHDL is the built-in simulation capabilities. VHDL, unlike many other hardware description languages and schematic capture programs, defines an integrated time reference. Device models can easily specify timing characteristics such as propagation delays and can check for timing violations such as setup and hold times. Timing specifications rely on the existence of faithful device models, therefore accurate timing simulations of Network Element subsections using structural VHDL is contingent on the availability of device models.

A test bench in VHDL is a model of a test fixture that can be used to test the device being designed with VHDL. The test bench is also written in VHDL, so all the capabilities of VHDL are available for sophisticated error checking. The test bench provides a non-proprietary way of stimulating and monitoring a design in a simulator. While some simulators provide for direct stimulus of a design without using a test bench, this capability is simulator dependent and is therefore not guaranteed to be present in all VHDL simulators. Even if the capability is included, the implementations may define different methods for specifying stimulus.

The test bench will be used for simulating major sections of the Network Element. Test benches will be developed to test the scoreboard, the fault-tolerant clock, the data path

voter, the global controller, and the ring buffer manager. A test bench will also be designed to test the aggregate of these subsystems in a complete Network Element design.

10.2.3. Testing

The VHDL models for the AFTA design will include test benches with which the models can be tested. A properly designed test bench can be used for either behavioral or structural VHDL models. In addition, the test bench can be used as a source of functional test vectors. The stimulus driven by the test bench and the expected response can be intercepted and saved in a data file. The data file can be used by the manufacturer of a submodule to test the submodule following assembly.

The test vectors derived from the test bench are functional test vectors which are intended to test that the device performs the desired function as defined in the original specification. An additional set of test vectors can be generated to test that a particular implementation of a device is free of faults. These test vectors are typically developed by automatic test pattern generator (ATPG) tools and test a device against the gate-level design description, not against the original design specification. Each set of test vectors should be used to fully test a device following fabrication.

10.2.4. Documentation

Another function of VHDL is to document the AFTA design. Documentation is required to enable reprourement or reimplementaion of the AFTA design.

10.2.4.1. Custom Devices

One of the most important applications of VHDL for documentation is for reprourement of custom devices used in the AFTA design. To maintain vendor independence for custom parts, a non-proprietary method is needed to unambiguously define the complete design of custom devices.

Many different options are available for reprourement of custom devices. These options are illustrated in Figure 10-5. The path chosen depends on the needs of the reprourement. If replacement parts for an existing system are needed, one of the paths with the least amount of effort should be sufficient. If new parts with architectural enhancements are needed, more effort will have to be expended to redesign many of the lower levels. The grayed areas indicate levels of the design to which VHDL can be applied.

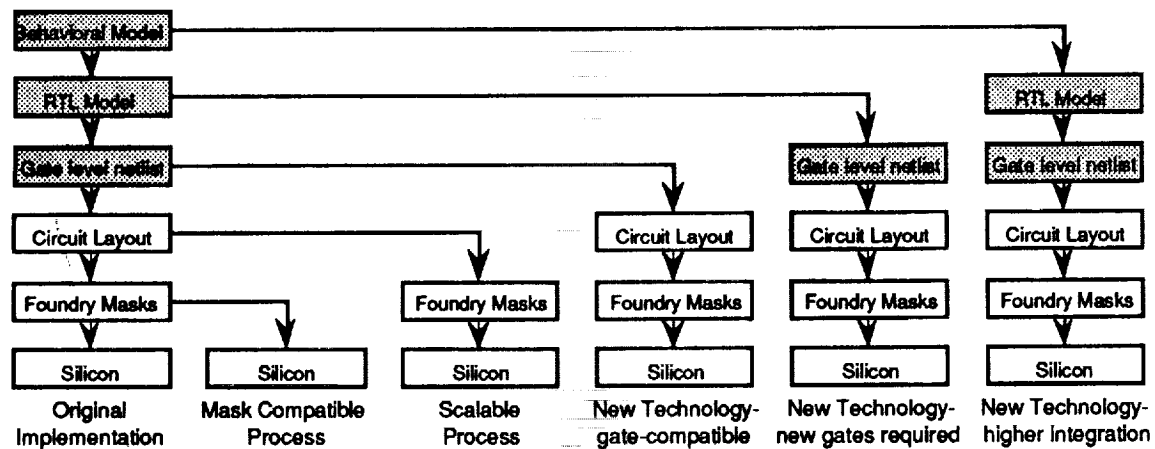


Figure 10-5. Reprourement Options for Custom Devices

The mask compatible process assumes that a standard, vendor independent process specification is used to fabricate an integrated circuit. Such a process would allow multiple vendors to reuse masks produced during the original fabrication process.

A scalable process is a derivative of an existing process in which the feature size, and thus the overall chip size, is reduced. If the design rules for the technology are scaled linearly with the feature size, a new chip can be fabricated from the original chip layout. New masks will have to be made, but the effort to produce masks from the original chip layout is a straightforward process.

Both of the above options are straightforward reprourement cycles. Little, if any, redesign is necessary. For these options, VHDL has no application. However, for reprourement cycles with more redesign effort, VHDL can be very useful.

The next level of reprourement is the use of a new technology with a compatible set of gates. For example, most CMOS, NMOS, GaAs, and TTL technologies use negative logic in the form of NAND and NOR gates. A gate-level netlist in VHDL that specifies instantiations of these gates could be used to port a design to any of these technologies. The VHDL behavioral model specifies the timing requirements for the design. The designer must make sure that the new gate-level netlist in the new technology meets the requirements specified by the original VHDL behavioral model.

Some alternate technologies use different types of gates. An example is ECL technology, which is based on OR/NOR gates. If a device that was originally designed for CMOS technology is to be reimplemented in ECL, a new gate-level netlist must be synthesized from the original register-transfer level (RTL) description to make effective use of the

OR/NOR gates in ECL technology. The RTL description, which could be written in VHDL, specifies blocks at a functional level, either as combinational logic or registers. The synthesis tool that creates the gate-level netlist from the RTL description uses a library that specifies what gates are available in the selected technology. Thus, the RTL description is the first truly technology and vendor independent specification level.

A final reprourement option is to reimplement a device at a higher level of integration. Increasing the integration level of a technology allows the designer to either produce smaller chips or to design chips that use more concurrent hardware for higher throughput. Architectural enhancements are a common method of improving the performance of a device without increasing the clock speed. However, to make these architectural enhancements, the designer must use the original high level specification for the device. The unpartitioned behavioral VHDL model is such a specification. The designer can use the behavioral model to determine what architectural changes can be made to optimize performance. Once the design is repartitioned with the architectural modifications, the rest of the design cycle must be completed before the chip can be fabricated in silicon.

The options presented above describe several different anticipated options for reprourement of custom devices for the AFTA Network Element. Reprourement can never be effortless, but the amount of effort can be made commensurate with the amount of change required.

10.2.4.2. Standard Devices

The use of VHDL is not limited to custom devices. VHDL can be used to specify a complete chip, board, or system level design in a hierarchical manner. VHDL is sufficiently versatile that a high level model of a system can be decomposed into board level and chip level components. Chip level components corresponding to custom devices can be decomposed further into gate level descriptions. Standard devices are described at the leaf-level by behavioral models defining the functionality and timing requirements of the devices.

Reprocurement of standard devices must be ensured as is the reprourement of custom devices. Typically, devices to be reprocured are required to conform to standard parts specifications defined in MIL-M-38510. Each of these devices is defined by a "slash sheet" in a standard, non-proprietary format. Each device is furthermore required to be available from more than one vendor. Each vendor must ensure that parts to be sold as compliant with the "slash sheet" meet the specification.

The reprourement options for systems containing standard devices are described in Figure 10-6. The options are much more limited than for custom devices. The reason for this is that either one can obtain an exact replacement for a part, or the design must be modified to accept a new part. The extent of the redesign is, of course, dependent on the relative similarity between the old part and the new part. However, redesign, even if simply a revalidation of the timing characteristics, will always be required if different parts are to be used.

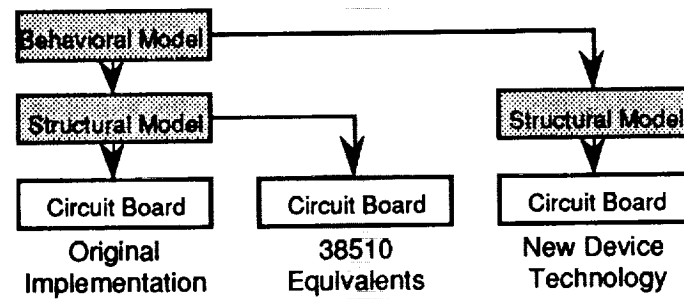


Figure 10-6. Reprourement Options for Standard Devices

The structural models represent the interconnection of individual standard devices. The structural model will not represent anything more than the netlist unless accurate behavioral models for the standard devices are used. If accurate models are not available, the netlist representation in VHDL has no advantage over more conventional netlist representations, such as schematic drawings.

The advantages to using VHDL for reprourement of standard devices is very marginal. Certainly, the behavioral VHDL allows for repartitioning of the design if new devices are to be used. However, the structural VHDL does not enhance the simple reprourement of equivalent devices.

The current approach to VHDL modeling for the AFTA design does not consider the leaf-level behavioral models for standard devices. The reason for this is the unavailability of these leaf-level models. Modeling devices at the behavioral level involves complex programming in VHDL and often requires significant knowledge of the inner workings of the device. In addition, these models have no use for the development of custom devices. Since the AFTA Network Element is targeted for multiple custom devices during full-scale development, the development of complex structural descriptions of standard devices seems to be fruitless.

10.2.5. Candidate VHDL Tools for AFTA NE Design

Draper has VHDL design tools and computing platforms in house to perform the above design functions. The available tools include products by Vantage, Viewlogic, and Synopsys. The Vantage and Viewlogic tools will be used for NE design capture and simulation. The Synopsys tool will be used to synthesize gate level netlists from the RTL VHDL description.

10.2.6. Compliance with Data Item Description

The data item description (DID) entitled "VHSIC Hardware Description Language (VHDL) Documentation," number DI-EGDS-80811 [DID80811], has some relevance to the development of VHDL models for the AFTA design. While the scope of the DID is more appropriate for a full-scale development project, certain sections of the DID will be addressed by the AFTA brassboard design to minimize the impact of future full-scale development of the AFTA design. Proposed compliance with, and deviations from, the DID are detailed in the following sections.

10.2.6.1. Reference Documents

The VHDL language and environment is defined by the IEEE Standard VHDL Language Reference Manual [IEEE1076]. The use of VHDL for the AFTA design will conform to [IEEE1076] in all aspects.

10.2.6.2. VHDL Model Hierarchy

A VHDL model of the Network Element design and of the subsections of the Network Element design will be provided. The Network Element is the only section that will be described using a structural model, unless detailed leaf-level models are provided by external means for either custom (ASIC) or standard logic devices. The subsections of the Network Element that will be described by VHDL behavioral models include the scoreboard, the fault-tolerant clock, the data path voter, the global controller, and the ring buffer manager.

10.2.6.3. Leaf-Level Modules

The leaf-level modules for the AFTA Network Element depends greatly on the targeted technology and the availability of accurate device models from an outside vendor. Development of these models is an expensive, time-consuming process and is outside the scope of the AFTA brassboard development project.

The Government may supply models for use in the AFTA design. These models will be incorporated as leaf-level modules wherever appropriate.

Major subsections for which a complete suite of models is not available will be supplied as either a behavioral model or an incompletely specified structural model.

A list of commercial grade parts used in the AFTA brassboard design will be provided along with the Network Element VHDL description. This list can be used to obtain appropriate device models at a later date.

10.2.6.4. Entity Declarations

The entity declarations for Network Element subsections will describe, as best as possible, the timing constraints of the subsection. Since the actual timing constraints depend on the behavior of the selected devices, the accuracy of these constraints is highly dependent on the availability of accurate leaf-level models.

The entity declarations for the Network Element subsections will conform to the interface declaration requirement as specified in the DID wherever possible.

The timing and electrical requirements for the Network Element design will not be guaranteed by the behavioral models of the NE. Any timing and/or electrical specifications included in the behavioral models will be derived from data book information. The structural models for the NE, if written, will include timing and electrical requirements if the models that make up the leaf-levels of the structural models handle these requirements properly.

The operating conditions for the Network Element design will not be handled by the behavioral models of the NE. The structural models for the NE, if written, will include operating conditions if the models that make up the leaf-levels of the structural models handle these requirements properly.

The high-level structural Network Element description will provide for the addition of timing, electrical, and operating condition requirements if appropriate leaf-level models become available.

The naming conventions for entities in the Network Element design will conform to the requirements as specified in the DID wherever possible.

10.2.6.5. Behavioral Body

Behavioral models for each major Network Element subsection will be developed as part of the AFTA detailed design process. The behavioral models will conform to the requirements as outlined in the DID wherever possible. Timing characteristics of the behavioral models may be based on preliminary analysis and may not reflect the exact timing characteristics of the brassboard design unless accurate leaf-level models are used to analyze the structural design.

Behavioral bodies will not be structurally decomposed unless functional partitions dictate that such decomposition is appropriate.

The timing characteristics of the behavioral body will specify, as accurately as possible, the known timing behavior of Network Element sections. Best, worst, and nominal output delays will be included, if known. However, many models (particularly those from commercial-grade data books) will only define worst case timing.

10.2.6.6. Structural Body

Structural models for major Network Element subsections will be developed as part of the AFTA detailed design process if appropriate leaf-level modules are available to define the structural model. The structural models will conform to the requirements as outlined in the DID wherever possible. The use of the structural models for logic fault modeling and test vector generation depends on the accuracy of the models instantiated in the structural models.

The naming conventions for components and signals in the Network Element structural design will conform to the requirements as specified in the DID wherever possible.

10.2.6.7. VHDL Simulation Support

The Network Element design will incorporate test benches for simulating the Network Element as a whole and for simulating major subsections of the Network Element alone. VHDL test benches will be written to be independent of a particular simulator product.

Each test bench will instantiate the appropriate module, either behavioral or structural, apply stimuli to the module's inputs, and test the module's outputs against an expected result. Any discrepancies in actual and expected output will be reported to the simulator op-

erator. Each test bench will incorporate a configuration to allow selection of the architecture for the model under test.

A test bench will be developed to test the entire Network Element as a stand-alone module. In addition, test benches will be developed to test the major subsections of the Network Element, including the scoreboard, the fault-tolerant clock, the data path voter, the global controller, and the ring buffer manager as stand-alone modules. Test benches for lower level entities will not be provided.

10.2.6.8. Error Messages

The format of error messages in the Network Element design will conform to the requirements as specified in the DID.

10.2.6.9. Annotations

Annotations included in the Network Element design will conform to the requirements as specified in the DID.

10.2.6.10. Reference to Origin

The models used in the Network Element design will specify the origin of the model as specified by the DID wherever appropriate.

The Government may supply models for use in the AFTA design. These models may be purchased by the Government for use in modeling the AFTA design, or provided from internal Government sources. The origin of these models will be specified, if known.

10.2.6.11. VHDL Documentation Format

The format of VHDL documentation for the Network Element design will conform to the requirements as specified in the DID wherever appropriate.

A facility for producing an ASCII tape under the specified requirements is currently in place. The organization of files on the tape will conform to the organization described in the DID. Not all files specified in the DID will necessarily apply to the AFTA brassboard VHDL models.

This page intentionally left blank.

11. AFTA Validation and Verification

AFTA will be used to provide mission- and vehicle-critical services in a range of Army applications. Some means must be defined to provide a reasonable degree of assurance that a given AFTA implementation will in fact perform the required mission functions: these means include the validation and verification processes.

Validation refers to the process of demonstrating that an implemented system correctly performs its intended functions, e.g., helicopter TF/TA/NOE/FCS, under all reasonably anticipated operational scenarios, fault conditions, computational loads, etc. Thus, what is ultimately wanted is a "validated system," about which one can state "This computer can perform helicopter TF/TA/NOE/FCS." To attempt to capture the relevant characteristics of a system which is believed to be capable of performing a mission's intended functions, a system specification is written with which the implemented system must comply, with the hope that a system meeting the specification will also perform the mission's intended functions[†]. Given a well-written specification, one can typically come close to building a system that meets it. However, human fallibility and the ambiguities of language – both informal and otherwise – conspire to ensure that no specification can completely describe the needs of a mission. In recognition of this, lengthy and expensive validation testing is typically performed on an implemented system, in which it is demonstrated that the system can in fact perform a specified set of the mission's intended functions over a specified envelope of operational conditions. The system is then called validated.

Given a specification, say of delivered throughput, the verification process demonstrates that an implemented system meets the specification. For the helicopter TF/TA/NOE/FCS example, it is perhaps desired to make the statement "This computer delivers $X_{VG,delivered}$ DAIS MIPS to the application program." The belief that a delivered throughput of $X_{VG,delivered}$ DAIS MIPS is sufficient to perform helicopter TF/TA/NOE/FCS is a critical link between successful performance of the mission's intended function and compliance with the system specification. Verification is relatively more straightforward than proving that an implemented system performs the mission's intended functions, because, whereas a mission environment incorporates the innumerable

[†] In actuality a hierarchy of specifications and requirements is written. For the purpose of the present discussion it is assumed that a single specification document suffices to describe the system.

vagaries, uncertainties, and complexities of real life, a specification is ultimately a list of requirements that can be enumerated and checked off during the verification process. The computer's delivered DAIS MIPS throughput can be benchmarked. In rare cases, a system specification is sufficiently exhaustive and accurate such that a system which meets the system's formal specification document is capable of performing the system's intended function; this assertion itself must of course be proved. In these cases, it is only necessary to verify that the system as implemented meets the system's formal specifications; validation is proved by the presumed logical transitivity between the specifications and the mission's intended function. Even in this idealized example, it is safe to say that the empirical phase of validation will never be eliminated; nobody would want to fly in an aircraft which had never been flight tested but which, we are assured, can be formally proven to meet its specifications.

Their limitations notwithstanding, specifications are written which serve as a mutually understood representation of the mission designer's understanding of what characteristics the computer system must have to perform the mission's intended functions, and the computer designer's understanding of what requirements the computer system must meet.

Three categories of statements can be made about a system. The first category contains statements (either regarding the mission's intended function or a specification item) whose validity can only be affirmed or negated via empirical test and evaluation for each and every differing implementation. The MTBF of an AFTA LRM is one example of such a statement: as the LRM changes due to technology insertion, or as the operational environment changes, the LRM's predicted MTBF must be verified via a reliability evaluation plan. While often unavoidable, use of such statements should be minimized since they comprise a large contribution to the cost of validating and verifying a system, and do not appropriately leverage experience gained from the implementation of prior systems. The latter two categories are indicative of a system which is described by the seeming oxymoron "validated independent of the application."

The second category contains logical statements (either regarding the mission's intended function or a specification item) which can be formally stated and shown to be an intrinsic property of the system when designed according to simple, unambiguous rules and guidelines. For example, one such statement refers to AFTA's Byzantine Resilience. Such statements have value because their relationship to these rules and guidelines has only to be shown once; subsequent implementations have only to be inspected to ensure that they comply with the rules and guidelines, and hence the statement about the system holds.

The third category contains quantitative statements (either regarding the mission's intended function or a specification item) describing certain important characteristics of a system that are valid, to an extent independent of the application in which that system is used. These characteristics are not usually fixed numerical quantifications of system attributes such as performance, reliability, component MTBF, etc., because these attributes change from implementation to implementation, as technology insertions occur, etc. Instead, implementation-invariant formulations are much more valuable in facilitating the cost-effective, safe, and predictable reuse of AFTA for various applications. One example might be the AFTA temporal overhead due to fault tolerance, surely an important determinant of delivered throughput. While it is tempting to state that fault tolerance consumes some fixed and attractively low temporal overhead based on prior experience, it is in fact the case that the fault tolerance-related overhead is a function of how often the application program invokes fault tolerance-related functions such as voting, synchronization, interactive consistency, etc. Thus it is critical to know the relationship between the application's invocation of these functions and the loss of throughput to allow the intelligent and informed design and partitioning of application tasks in AFTA and the accurate prediction of the actual delivered throughput in an implementation.

In reality, general statements about AFTA are supported by statements from all three categories described above. For example, a statement about AFTA reliability includes statements about LRM MTBFs (only available via empirical test and evaluation for each LRM type during Reliability Development and Growth Testing), Byzantine Resilience (a logical attribute of AFTA which can be shown via adherence to simple architectural rules), and fault latency (which is partly a function of the frequency at which the FDIR/C-BIT task is executed on a VG).

In summary, validation is the process of demonstrating, with a high degree of confidence, that a system correctly performs its intended mission functions, for example, helicopter flight control. Verification is the process of demonstrating that a system meets its specifications, for example, 10 MIPS throughput. A verified system becomes a validated system when a correspondence has been shown between the specifications and the intended mission functions, for example that 10 MIPS is sufficient throughput to perform helicopter flight control. Application dependent specifications, such as throughput, cause validation process to be repeated for each new application. However, there are certain logical statements that can be made about AFTA attributes that hold true independent of applications, such as Byzantine Resilience. In that sense, once these attributes of AFTA have been

demonstrated, one can say that AFTA is partially validated independent of the intended application requirements.

11.1. Verifiable AFTA Attributes

The following list shows AFTA attributes which will be verified during the Dem/Val phase. This set of attributes can be expanded upon guidance from the Army.

<u>Functional Correctness and Byzantine Resilience</u>
Fault Containment
NE Synchronization
Interactive Consistency
Voting
Message-Release Authorization (Scoreboard)
Functional Synchronization
Byzantine Resilient Virtual Circuit Abstraction
Reconfigurability
Rate Group Scheduling
Intertask Communication Services
I/O Services
Redundancy Management (FDIR) Software
<u>Performance-related Attributes</u>
Delivered throughput
Available memory per VG
Effective intertask communication bandwidth
Effective Input/Output bandwidth
Task iteration rate
<u>Reliability-related Attributes</u>
AFTA reliability and availability
<u>Cost-related Attributes</u>
Cost per Unit of Service
<u>Physical Attributes</u>
Weight
Power
Volume

Table 11-1. Verifiable AFTA Attributes

If this list seems somewhat short it should be borne in mind that each attribute listed above must be plausibly verified within a reasonable cost and time; therefore it makes sense

to keep the size of the list to the minimum needed to specify the core AFTA attributes needed to perform the mission's intended function.

The AFTA Conceptual Study focuses on "predictive" verification, in which AFTA is verified using a combination of predictive and corroborative verification means. First, the AFTA's verifiable attributes are enumerated; these correspond to the requirements definition format defined in Section 2 of this report. In the predictive phase of the verification, these AFTA attributes are predicted via performance, reliability, availability models, and cost models as described in Section 9. The predictive phase runs through the Conceptual Study and Detailed Design phases of the AFTA program. In the corroborative phase, executed during the Brassboard Fabrication, Integration, and Validation phase of the AFTA program, critical model inputs are verified via empirical test and evaluation, or sensitivity studies are performed to obtain bounds on the effects of unverifiable parameters. In addition, quantities predicted by the models which can be empirically verified are measured to corroborate the models' accuracy.

In empirical verification it is necessary that either (a) sufficiently large sample sizes must be obtained to produce statistically sound results to support the modeling assertions or (b) the architecture must be designed to minimize reliance on assertions which rely on statistical parameters which can not be obtained with a reasonable amount of effort. The AFTA design is intended to be strongly biased towards the latter approach.

11.2. Verification of Byzantine Resilience and Operational Correctness

Numerous functions must be performed correctly for AFTA to meet its advertised performance and reliability requirements. These functions may be expressed as a set of logical statements about the arrangement and operation of the architecture which are independent of any application in which AFTA is used. Thus, if correctly implemented, these statements may be viewed as being valid regardless of any application.

The first set of specifications describes architectural features which are required for AFTA to be a Byzantine resilient message-passing parallel processor; these features are largely implemented in the Network Element. The second set of specifications describes the functionality required for the AFTA OS to successfully perform scheduling, message passing, I/O, and redundancy management.

11.2.1. Fault Containment

AFTA must be partitioned into at least four Fault Containment Regions (FCRs), each of which contains at least one NE. The FCRs must possess independent sources of power, independent clocks, be dielectrically isolated from each other, and if damage tolerance is required, be physically separated from each other. Verification of each of these requirements is performed by inspection of the AFTA design and implementation.

11.2.2. NE Synchronization

The AFTA Network Elements must be synchronized to each other to within a known skew. The NEs achieve synchronization via the use of a set of circuitry known as the Fault Tolerant Clock (FTC), which executes a Byzantine resilient phase locking algorithm. The synchronization algorithm itself must first be specified and shown to be Byzantine resilient. This has been done via journal-style mathematical proof in [Kri85].

The circuitry must then be shown to correctly implement the algorithm. It is preferable that this be done via the process of formal specification and verification. In fact, because the NE is a critical component of AFTA it is highly recommended that its entire functionality be subjected to a well-supported program of formal specification and verification. This is feasible with the current state of formal verification technology but is inappropriate within the limited AFTA Brassboard Dem/Val schedule and budget. Therefore the approach taken in the Brassboard Dem/Val phase is to design the FTC and other NE hardware according to standard engineering practice, including detailed specification, design, implementation, and test reviews, with the intent that eventually formal methods will be applied to this circuitry. Under the Brassboard Dem/Val phase, the NE synchronization skew will be measured in the absence of faults and in the presence of Byzantine faults.

11.2.3. Interactive Consistency

Distribution of data from one member to all members of a redundant VG must be performed using a Byzantine resilient interactive consistency algorithm. The algorithm used in AFTA has been formally specified and demonstrated to be Byzantine resilient in [LSP82]. The circuitry must be shown to correctly implement the algorithm. The preferred approach to verification of the correctness of the AFTA interactive consistency function is via the process of formal specification and verification. Again, because of schedule and budget constraints, a more traditional hardware engineering process will have to do during

Dem/Val. The interactive consistency circuitry will be empirically shown to correctly implement the algorithm for all possible data sources and all possible data destinations, in the presence of one Byzantine fault.

11.2.4. Voting

Messages emanating from redundant VGs are passed through a majority vote function implemented in the NEs. The NEs must be capable of voting messages arriving from triplex and quadruplex VGs. The voter is maskable, and generates vote syndromes for delivery to the destination VG. Voting is easily expressed mathematically and demonstration of its Byzantine resilience is relatively straightforward. The NE's voting circuitry must be shown to correctly implement the algorithm, and the above comments regarding formal specification and verification hold. The approach taken in the Dem/Val phase is to design the voter according to standard engineering practice. The voter will be empirically shown to correctly vote input messages. In the presence of faulty input messages, the voter will be shown to correctly generate a syndrome corresponding to the faulty input. The voter will be shown to have the capability to mask out an input such that it can not contribute to the voted outcome. The response of the voter to out-of-specification inputs, such as "two-two splits," will be demonstrated.

11.2.5. Message-Release Authorization (Scoreboard)

The NE Scoreboard has responsibility for deciding which inter-VG messages may be transmitted. This calculation is a complicated function of the message request pattern, the flow control request pattern, the redundancy configuration of AFTA, the sender's and receiver's redundancy levels, and elapsed time. The baseline approach to specification and verification of the Brassboard Scoreboard is, again, standard engineering practice. The Scoreboard will be shown to generate correct message release decisions from a large set of input message request patterns, where the patterns will in many cases be those resulting from faulty source and destination PEs.

However, because of its complexity and criticality, the Scoreboard has been targeted for more advanced specification, design, and verification approaches. If these approaches are cost-effective they may be used for other parts of the NE. The Scoreboard algorithm has been expressed in the C programming language and by a VHDL description. These C and VHDL descriptions have been stimulated with representative message request patterns to help in identifying errors in the specification. The patterns serve as a verification suite

for successively detailed representations of the Scoreboard. VHDL will be used in conjunction with ASIC synthesis tools to automatically transform the high-level behavioral description of the Scoreboard to an implementation. Verification of the fidelity of the Scoreboard implementation to the behavioral specification will be performed by applying the message pattern verification suite to the representation of the detailed implementation. This is facilitated by the ability of automated VHDL synthesis tools to provide mechanical traceability between the hierarchical representations of the design, all the way from the high-level behavioral description to the transistor level.

To investigate the feasibility of formal specification and verification of AFTA hardware, the VHDL and textual description of the Scoreboard is being transformed into a formal specification under a collaborative effort with Odyssey Research Associates. It is expected that the expression of the Scoreboard algorithm in a rigorous formalism will assist substantially in revealing incompleteness, ambiguity, and inconsistency in the specification. It will also serve as a concrete case study for estimation of the time and effort in constructing formal specifications and design verifications of other parts of the NE.

11.2.6. Reconfigurability

The mapping of PEs to VGs in AFTA may change upon command from a VG having an appropriate redundancy level. It is in fact this capability to reconfigure processing resources in real time which gives AFTA its power to provide high reliability and availability across a wide variety of missions and mission modes. The mapping of PEs to VGs is effected through a Configuration Table (CT) resident in the NEs. The CT is used by the Scoreboard to interpret message request patterns and determine the sources and destinations of messages. The CT in the NE is changed upon reception of a command known as a "CT Update" emanating from an appropriate VG. It must be verified that, given a CT, a given message request pattern results in selection of the appropriate message source and destination. This is subsumed in the above discussion on verification of the Scoreboard correctness. Next, it must be verified that a given CT Update in fact causes the Scoreboard to correctly reconfigure the mapping from physical (PE) to virtual (VG) resources according to the CT Update's contents. This rests on verification of two statements. First, it must be verified that the CT Update emanating from the VG is correctly voted and delivered to the NEs. This is subsumed in the verification of the correctness of the NE's voting function. Second, it must be verified that the scoreboard receives the voted CT Update and correctly updates its CT. Verification of this attribute is also subsumed in the Scoreboard verification effort.

11.2.7. Functional Synchronization

The members of a redundant AFTA VG are synchronized via the synchronous reception of copies of a message, as described in Section 3. To perform functional synchronization, the members of a VG transmit a message to themselves and await its reception. The synchronized NEs perform identical computations on the message request pattern, vote or perform interactive consistency on the message, and deliver the message to the destination VG with a small skew. To verify that functional synchronization in fact synchronizes the VG members it is necessary to verify that, at synchronization points determined by the AFTA OS, the VG members transmit a message and await its reception. This is done via examination of the OS code which purports to achieve functional synchronization. It is next necessary to verify that the NEs are synchronized, that they perform identical computation on the message request pattern (via the Scoreboard), that they can vote or perform interactive consistency on the message, and that they select the correct sources and destinations for the message. Means for verifying these assertions are enumerated above. Functional synchronization will be demonstrated both in the presence and absence of PE and NE faults. The time required for synchronization and the post-synchronization skew will be measured.

11.2.8. Byzantine Resilient Virtual Circuit Abstraction

The BRVC, described in Section 3, is the major inter-VG communication abstraction provided by the NEs comprising the fault tolerant core of AFTA. All higher-level AFTA OS functionality relies upon this abstraction for message ordering, correctness, and delivery skew. The BRVC comprises the following Byzantine resilient guarantees:

Guarantee 1: Messages sent by non-faulty members of a redundant source VG are correctly delivered to the non-faulty members of recipient VGs.

Verification: This guarantee is verified by demonstrating that the NE aggregate can correctly vote messages (Section 11.2.4) and route them from their source to destination VG (Section 11.2.6).

Guarantee 2. Non-faulty members of recipient VGs receive messages in the order sent by the non-faulty members of the source VG.

Verification: This guarantee is verified by demonstrating that the NE Scoreboard releases a message emanating from a source VG before releasing any others from that same VG (Section 11.2.6).

Guarantee 3. Non-faulty members of recipient VGs receive messages in identical order.

Verification: This guarantee is verified by demonstrating that all NEs achieve interactive consistency on the message request pattern (Section 11.2.3), and all Scoreboards correctly execute the same message release algorithm (Section 11.2.6).

Guarantee 4. The absolute times of arrival of corresponding messages at the members of recipient VGs differ by a known upper bound.

Verification: This guarantee is verified by demonstrating that all NEs are synchronized (Section 11.2.2).

11.2.9. Rate Group Scheduling

The next set of logical statements refer to the AFTA OS. Under the Detailed Design phase of the program, a Software Development Plan (SDP) and Software Requirements Specification (SRS) will be constructed for the AFTA OS. The SRS will describe the functionality of each of the major AFTA OS functions as well as qualification, acceptance, and verification tests for each. The discussion presented below represents a high-level overview of the most important functions achieved by the OS and the means for their verification.

At the highest level of abstraction, the Rate Group dispatcher is responsible for starting tasks belonging to a given Rate Group (RG) at the beginning of that RG. Each minor frame demarcates a specified set of RGs; tasks in these RGs must have finished one iteration and are prepared to begin their next iteration. The following table illustrating the RG boundaries is reproduced from Section 9.

Frame Boundary	Completed RGs	Started RGs
7-0	4, 3, 2, 1	4, 3, 2, 1
0-1	4	4
1-2	4, 3	4, 3
2-3	4	4
3-4	4, 3, 2	4, 3, 2
4-5	4	4
5-6	4, 3	4, 3
6-7	4	4

Table 11-2. Completed/Started RGs vs. Minor Frame Boundary

It must be verified that, on the appropriate frame boundary, the RG dispatcher enables execution of the tasks whose RG frames are about to begin via setting an event corresponding to each RG; RG tasks which have just completed their frames are awaiting the setting of this event to resume execution. To verify that the appropriate RG tasks are started on the appropriate RG frame boundaries, it must be shown that the RG dispatcher sets the appropriate events upon each minor frame boundary. In addition, it must be shown that each RG task awaits the setting of the event corresponding to its RG frame upon completion of each of its iterations. Formal specification and verification of this and other selected AFTA OS functionality should be performed; while somewhat costly and difficult given the current state of the art, the criticality of the OS functionality to the AFTA's reliable operation merits this effort. Because of cost constraints in the AFTA Brassboard phase, verification of the correct implementation of the OS functionality will be done via standard software engineering practice, which includes detailed specification, documentation, review, and exhaustive testing of the OS and application interface code. In the testing phase, it will be demonstrated that the RG dispatcher correctly dispatches a specified number of tasks in each RG, ranging from zero to the maximum number specified by the dispatcher specification. Tasks shall be constructed which test the error handling capabilities of the dispatcher. For example, such tasks shall generate frame overruns in order to test the dispatcher frame overrun detection and recovery capability, generate Ada exceptions which have no handler and hence trap to the dispatcher handler, and attempt other malfeasance.

11.2.10. Intertask Communication Services

AFTA tasks communicate using the intertask communication services described in Section 5. The services consist of four main components. The message enqueueing component receives a message from an application task, partitions it into Network Element packets, and places these packets on the message queue corresponding to the RG hosting the application task. At each minor frame boundary, the packet transmission component transmits the queues of packets corresponding to just-completed RGs into the Network Element for transmission to the destination VG(s). The packet delivery interrupt service routine fields packet delivery interrupts from the Network Element by copying the delivered packet to the appropriate incoming packet queue. Finally, the message reception component updates frame markers, constructs completed messages from the incoming packet queue, and makes these messages available to destination tasks. Each of these functions must be specified and verified in detail. Formal methods are again recommended but because of their cost will not be utilized during the Brassboard construction. Again, standard software development practice will be used, which will include a rigorous testing program in the presence of faults. During the testing phase, it will be demonstrated that the intertask communication services correctly transmit messages for all message sizes up to the maximum allowable in the intertask communication specification, for all exchange classes, and for all source and destination task combinations including broadcasts. Tasks shall be constructed which test the error handling capabilities of the communication services. Such tasks shall attempt to overflow their transmit buffers to test outgoing flow control capability, cease reading incoming messages to test the incoming flow control and buffer overrun containment features of the communication services, attempt to cause flow control at selected destination VGs by sending many large messages to it, send messages to nonexistent tasks and VGs, send illegal message classes, send illegal message sizes, and exhibit other erroneous behavior.

11.2.11. I/O System Services

The AFTA I/O System Services (IOSS) are composed of four components. The I/O dispatcher ensures that I/O requests (IOR) are executed and processed on appropriate frames. The IOR execution component initiates concurrent and sequential I/O. The IOR processing component reads input and status data and delivers it to the destination task(s). The "back end" device drivers to which the IORs interface perform the detailed bit- and byte-level manipulation of the interface to the IOCs. The correctness of each of these components must be verified. It must be shown that the I/O dispatcher invokes execution and

processing of each IOR in the frames in which it is specified. Each IOR must be shown to execute and process the correct I/O chains when it is scheduled, and interface to the correct back end I/O driver routines. These are algorithmic functions which lend themselves to formal specification and verification. While the baseline approach to specification and verification of the AFTA IOSS components follows standard software development practice, these services are also being targeted for the use of a formal software specification and verification tool; if the use of this tool proves cost-effective then it may be used on other AFTA software components. Finally, because of their highly specific and non-algorithmic nature, the back end I/O driver routines' correctness will be demonstrated via standard software development procedures, which will include extensive testing in the presence of faults. Tasks shall be constructed which stress the error handling capabilities of the IOSS.

11.2.12. Redundancy Management (FDIR) Software

The FDIR software is responsible for testing components, detecting and identifying faulty components, performing fault recovery actions, and performing reconfigurations as mission modes change. As can be seen from Section 5, the FDIR function can be extremely complex and require the collaboration of multiple VGs in a distributed, fault tolerant algorithm. Moreover, although much of FDIR is active in the absence of faults, the true utility of FDIR is in the presence of faults, since it determines how AFTA will respond to faulty behavior. These complexities require the use of a set of verification techniques. The FDIR functions relevant to a single VG may be specified in detail, possibly using formal methods, and it can be shown that the FDIR code as implemented correctly reflects its specification, either through standard software development means or formal means. This includes showing that the FDIR task, when scheduled by the RG dispatcher, correctly performs the intra-VG presence tests, and, if a member is faulty, correctly identifies the faulty member and performs the selected local recovery procedure. AFTA-wide fault detection, diagnosis, and recovery actions may also be specified and analytically verified, but analytically showing the correctness of implementation in the presence of faults, especially of the recovery functions, may be very difficult because the loose synchronization between multiple VG participants adds temporal complications.

In addition to analytical means, FDIR must at least in part be verified by a process of empirically examining its response to faulty behavior. Faults may be injected in software (e.g., FIAT, DEPEND) or hardware (e.g., AIPS, FTMP), each of which has its merits and will be used judiciously. For each fault recovery policy to be used in an implementation, it is necessary to fault each LRM while AFTA is in every possible configuration of redundant

VGs and IOCs, and confirm that the fault is detected and the designated recovery policy is carried out. Timing data will also be obtained. Empirical test and evaluation is particularly important in verifying multi-VG response to faults. The number of fault and configuration combinations in a testing program such as this is admittedly large but it is felt that the FDIR function is of sufficient importance that the effort be made. Certain simplifications can be made to make the testing program tractable, such as injecting errors only at the LRM level, and limiting the temporal behavior of faults to permanents and transients. Automating the fault/error injection and data acquisition processes will be necessary.

11.3. Verification of Performance Predictions

Verification of performance predictions usually requires empirical measurement of the quantity of interest. The AFTA PEs' throughput(s) may be initially estimated via empirical benchmarking using Whetstones, Dhrystones, and other mutually agreed upon empirical benchmarks; alternatively, the throughput(s) may be analytically evaluated as in a DAIS mix calculation. Empirical timing measurement of existing AFTA components may be performed with the use of processor emulator pods, logic analyzers, processor-asserted discrete outputs to logic analyzers or oscilloscopes, or on-processor timers. All empirical evaluations must be done in the presence of worst-case faults such as a failure of a VG's processor or a Network Element. It should be borne in mind that sample sizes obtainable from empirical evaluation, while overwhelmingly large, may be insufficient to support statistically viable assertions that hard real-time constraints will be met with a probability commensurate with the reliability requirements of flight-critical systems, that is, on the order of 0.999,999,999. Therefore the AFTA's quantitative characteristics are intended to minimally rely on such assertions.

11.3.1. Delivered Throughput per VG

The delivered throughput per VG is defined to be the raw PE throughput minus operating system, redundancy management, and synchronization overheads. The delivered AFTA throughput is equal to the delivered throughput per VG times the number of VGs in AFTA. A model relating the raw throughput to the delivered throughput and various overheads is presented in Section 9. The following parameters of this model must be verified either via inspection of the design and application tasks, empirical measurement, or static calculation of upper-bounds on execution times as in [Pus89].

Parameter	Verification Technique
$X_{VG, raw}$, the raw VG throughput	Determined via benchmarking according to standard benchmarks; depends critically on computation characteristics
T_{HK} , the dispatcher housekeeping time	Empirical timing measurement on AFTA or calculation of execution upper-bound
$N_{MESSAGES, i}$, the number of messages sent in frame i	Empirical inspection of application tasks
T_{SU} , the setup time required to begin sending a single message	Empirical timing measurement on AFTA or calculation of execution upper-bound
S_k , the size (in Network Element packets) of outgoing message k in frame i	Empirical inspection of application tasks
T_p , the incremental time required to send one packet	Empirical timing measurement on AFTA or calculation of execution upper-bound
$N_{MESSAGES, i}$, the number of messages received in frame i^*	Empirical inspection of application tasks
T_{SU} , the setup time required to begin receiving a single message [*]	Empirical timing measurement on AFTA or calculation of execution upper-bound
S_k , the size (in Network Element packets) of incoming message k in frame i^*	Empirical inspection of application tasks
T_p , the incremental time required to deliver one packet [*]	Empirical timing measurement on AFTA or calculation of execution upper-bound
$N_{TASKS, i}$, the number of tasks to be started in frame i	Empirical inspection of application tasks
T_{EV} , the time required for the dispatcher to set an event	Empirical timing measurement on AFTA or calculation of execution upper-bound
$T_{FDI, minor}$, the time required for one minor frame execution of the FDI task	Empirical timing measurement on AFTA or calculation of execution upper-bound
T_{CS} , the context switch time per task	Empirical timing measurement on AFTA or calculation of execution upper-bound
$N_{TASKS, R4}$, the number of R4 tasks	Empirical inspection of application tasks
$N_{TASKS, R3}$, the number of R3 tasks	Empirical inspection of application tasks
$N_{TASKS, R2}$, the number of R2 tasks	Empirical inspection of application tasks
$N_{TASKS, R1}$, the number of R1 tasks	Empirical inspection of application tasks

Table 11-3. Verification of Delivered Throughput

* Name reused to avoid nomenclature proliferation.

11.3.2. Available Memory per VG

The available memory per VG is defined to be the gross memory per VG minus the Ada Run Time System, dispatcher, and FDI memory requirements. The total AFTA VG memory is defined to be the available memory per VG times number of VGs in AFTA.

The verification approach for this attribute is straightforward. The gross memory per VG is determined by inspection of the design of the PEs comprising the VG; this quantity is probably specified in the PE procurement specification. The Ada RTS, dispatcher, and FDI memory requirements for a given mission are empirically determined by inspection of the memory map of the load module of the VG of interest.

11.3.3. Effective Intertask Communication Bandwidth and Latency

The effective intertask communication bandwidth is the size (number of bytes) of an intertask message divided by the time required for transmission and reception of the message. The latency is the time between the transmission of the message by the sending task and the reception of the message by the recipient task. A model for the latency is presented in Section 9. The verification parameters needed to allow this model to accurately predict the effective intertask communication bandwidth and latency are listed in Table 11-4.

11.3.4. Effective I/O Bandwidth and Latency

The effective I/O bandwidth is defined to be the size (in number of bytes) of an I/O transaction divided by the time required for transmission (reception) of the transaction by the source (destination). The input latency is the time in seconds between the sampling of an input byte by the input device and the availability of that byte at the input of the destination function. The output latency is the time in seconds between when a computational function generates an output byte for delivery to an output device and when the output device receives the output byte.

Because the I/O devices to be used in AFTA are not yet definitized, a detailed list of parameters to be measured can not be constructed. The general outline presented in Table 11-5, however, seems appropriate for verification of the performance any type of I/O device and transaction type.

Parameter	Verification Technique
T _{ENQUEUE MESSAGE} , the time required for a task to enqueue a message for transmission	Empirical timing measurement on AFTA or calculation of execution upper-bound
T _{latency, RG} , the time between the enqueueing of a message by a sending task and the task's next RG frame boundary [†]	Empirical timing measurement on AFTA
T _{SU.XMIT} , the time required for the communication services to prepare a message for transmittal	Empirical timing measurement on AFTA or calculation of execution upper-bound
T _{XMIT} , the time required for a packet to be transferred from the PE to the NE	Empirical timing measurement on AFTA (straightforward upper-bound measurement)
T _{NE} , the time required for the network element ensemble to perform the requested message transmission	Empirical timing measurement on AFTA (straightforward upper-bound measurement)
T _{RECEIVE MESSAGE} , the time required for the communication services to construct incoming message from packet queue	Empirical timing measurement on AFTA or calculation of execution upper-bound

Table 11-4. Verification of Intertask Communication Bandwidth and Latency

Parameter	Verification Technique
Transaction size	Examination of application code
For memory-mapped I/O, time between transaction start and transaction processing completion	Empirical timing measurement on AFTA (straightforward upper-bound measurement)
For network output, time between transaction start and reception of data at output device	Empirical timing measurement on AFTA (straightforward upper-bound measurement)
For network input, time between transaction start and initiation of processing for transaction	Empirical timing measurement on AFTA (straightforward upper-bound measurement)
Time required for I/O task to process transaction information	Empirical timing measurement on AFTA or calculation of execution upper-bound
Time required for I/O task to deliver processed transaction information to local or remote destination task(s)	Subsumed under intertask communication bandwidth and latency verification

Table 11-5. Verification of I/O Communication Bandwidth and Latency

[†] The programming model is designed to be insensitive to this parameter.

11.3.5. Iteration Rate of a Task

The iteration rate of a task is defined to be the frequency at which task iterations are initiated, it being assumed that the execution time of a task iteration is less than the reciprocal of the iteration rate. The approach to verifying that a task's specified iteration rate will be serviced by AFTA begins with showing that the task is assigned to a Rate Group (RG) corresponding to its desired iteration rate. It must then be verified that the RG dispatcher initiates the RG at the requisite frequency. Verification of this functionality is subsumed in Section 11.2.9. All Rate Group tasks will execute at their desired iteration rate if the total throughput consumed by all RG tasks,

$$T_{RG} = f_{R4}X_{R4} + f_{R3}X_{R3} + f_{R2}X_{R2} + f_{R1}X_{R1} \quad (11.1)$$

is less than $X_{VG, \text{delivered}}$, as calculated in Section 9:

$$T_{RG} \leq X_{VG, \text{delivered}} \quad (11.2)$$

where

T_{RG} = total throughput consumed by all RG tasks

f_{Ri} = frame rate of RG i

X_{Ri} = throughput requirement of one iteration of all tasks in RG i

11.4. Verification of Reliability and Availability Predictions

The AFTA reliability, $R(t)$, is the probability that AFTA correctly performs its intended function during the time interval $(0,t)$, given that it was in a well-defined operational state at time $t=0$. The AFTA availability, $A(t)$, is the probability that the system is capable of performing its intended function at time t , with temporary outages during the time interval $(0,t)$ being allowed for repair.

As described in Section 9, several formulations can be used to quantify AFTA reliability and availability, depending on the redundancy management and fault recovery options in use. A general expression for AFTA reliability and availability is related to the probability that AFTA can perform its intended functions at time t , given a particular redundancy management option; this probability can correspond to AFTA reliability or AFTA availability, depending on the mission and/or mission phase that it models. Two representative formu-

lations for this probability are given in Section 9. $p_{GD}(t)$ represents the probability that AFTA can perform its intended functions at time t when the components are managed under a graceful degradation class of fault recovery policies, in which no service interruption is incurred. $p_{PR}(t)$ represents the probability that AFTA can perform its intended functions at time t when the components are managed under a processor replacement class of fault recovery policies, in which brief outages may be incurred for fault recovery. Both calculations assume that all AFTA components are operational at time $t=0$.

Since the reliability and availability of AFTA in its redundant configurations are far too high to verify using accelerated life testing, verification of AFTA reliability and availability must be performed indirectly through a combination of analytical and empirical means.

In the analytical phase, predictive models are constructed for the figure of merit of interest; the models in Section 9 are two such predictive models. The models rely on the Byzantine resilience of the underlying AFTA architecture; verification of this assumption is described in Sections 11.2.1 through 11.2.8. The models also rely on abstractions of AFTA behavior in the presence of faults; the detailed behavior of a particular fault recovery option is specified and verified in Section 11.2.12. The correspondence between the analytical model's abstraction of the behavior and the actual behavior must be verified by detailed comparison of the (preferably verified) FDIR implementation and the analytical model. This phase of reliability and availability verification is performed once and for all, since the validity of the models is independent of the application.

General classes of numerical inputs to the reliability and availability models are listed below. The numerical inputs may change from implementation to implementation and from mission to mission, and therefore have to be reverified for each application.

11.4.1. Component Failure Rate

Component failure rates are a first-order determinant of ultimate AFTA reliability and availability and must be empirically verified or computed in compliance with acceptable engineering practice. AFTA LRM/LRU failure rates are computed using the Parts Stress Analysis techniques as specified in MIL-STD-217E. It is assumed that the use of the MIL-STD-217E-based approach yields failure rate data which are reasonably accurate. In some cases failure rate data can be empirically corroborated through Reliability Growth/Development Testing and field data. In the AFTA analytical models constant fail-

ure rates are assumed. This assumption must be verified using acceptable engineering practices.

11.4.2. Fault Reconfiguration Time

Lengthy fault reconfiguration times can result in a significant probability of AFTA failure due to a second fault occurring while AFTA is recovering from the first. AFTA is designed to mitigate the effects of such near-coincident faults during the mission by using a rapid fault reconfiguration, i.e., reconfiguration within 10 ms of vote error manifestation. For anticipated AFTA mission durations, this short reconfiguration time can reduce the probability of VG failure due to near-coincident faults (the second term in Equation 9.23) to a level which is relatively small compared to the probability of VG failure due to attrition (the first term in Equation 9.23). Minimization of near-coincident faults as a dominant VG failure mode reduces the need for acquiring statistically significant measurements of the reconfiguration time, since the architecture's reliability and availability are designed to be insensitive to this quantity.

Table 11-6 illustrates a typical relationship between the two failure modes' probabilities for the helicopter mission. Note that the two contributors to VG failure are commensurate for quadruplex VGs being used for short mission times, and therefore statistically significant verification of reconfiguration time becomes an issue in this case. In general, large variations in the AFTA mission duration or reconfiguration time may force this quantity into prominence. The AFTA reliability and availability models compute the probability of failure due to near-coincident faults and attrition to allow estimation of their relative importance and the consequent need for intensive verification of reconfiguration time. In these models constant reconfiguration rates are assumed, which yield a pessimistic estimation of probability of failure due to near-coincident faults.

VG Redundancy Level	Probability of VG Failure due to Attrition	Probability of VG Failure due to Near-Coincident Faults
1 Hour Helicopter Mission		
Triplex	6.49E-09	7.21E-14
Quadruplex	7.14E-13	1.44E-13
2 Hour Helicopter Mission		
Triplex	2.59E-08	1.44E-13
Quadruplex	4.84E-12	2.88E-13

Table 11-6. VG Failure Probability Due to Attrition and Near-Coincident Faults

During the Brassboard Dem/Val phase, fault reconfiguration times will be empirically measured. It is expected that they will depend strongly on the reconfiguration policy in effect, the throughput of the PEs, and the bandwidth of the various communication components like the NEs and FCR backplane bus.

11.4.3. Fault Reconfiguration Coverage

The AFTA analytical models assume unity detection and reconfiguration coverage for faults occurring in triplex and quadruplex VGs, and quadruplex and quintuplex NEs. These high coverages are due to the architecture's compliance with the requirements of Byzantine resilience, and are not generally considered empirically verifiable per se. Verification of the correctness of the synchronization, voting, and FDIR functions that imply this unity coverage has been discussed above. An additional issue arises of verifying the coverage of faults occurring in degraded triplex VGs, commonly denoted the "duplex coverage." Three options exist, each of which impacts verification. First, the graceful degradation policy described in Section 5 can leapfrog the degraded triplex state and proceed directly to the simplex state. This transition can occur with unity coverage since a triplex can accurately diagnose its own health in the presence of a single fault and, specifically, can identify a nonfaulty survivor simplex within its own VG. A second option is to assume that a degraded triplex VG suffering an additional fault randomly determines which of its two members is nonfaulty and designates that member as the surviving simplex. This results in a duplex coverage which is probably about 0.50. Finally, self-tests can be developed which can increase the duplex coverage to something better than a random guess. For verification purposes, the first policy is superior since it introduces no additional verifiable parameters.

11.4.4. VG Redundancy Levels

Verification of the redundancy levels of the VGs comprising an AFTA configuration is straightforward.

11.4.5. Mission/Hiatus Time

Verification of the mission and hiatus times for an anticipated AFTA mission is straightforward. As an anticipated mission evolves the analyses must be continually repeated to ensure that the targeted AFTA configuration continues to meet the reliability, availability, and life cycle cost objectives.

11.5. Verification of Cost Predictions

The Fleet Life Cycle Cost per Service Unit (FLCCPSU) is the life cycle cost of a fleet of vehicles given a required sortie rate, including the cost of additional vehicles required due to vehicle unavailability, the cost of repairs and spares, the cost of redundancy, and the cost of vehicles lost due to vehicle unreliability, over the fleet service life. A simplified model for the FLCCPSU is given in Section 9. As for the reliability and availability, FLCCPSU can not be directly measured, but must instead be verified through a combined program of predictive analysis and empirical means.

A predictive model for FLCCPSU is submitted in Section 9 of this report. Verification of the adequacy of this model consists of review of the model by appropriate government representatives, followed by recommendations for improvement and subsequent concurrence that it represents FLCCPSU sufficiently well for the purposes of drawing conclusions regarding AFTA redundancy levels, fault recovery policies, and other architectural and operational parameters. The FLCCPSU model draws upon analytical results from other AFTA modeling efforts such as performance, reliability, availability, weight, power, and volume, and hence its validity rests upon theirs. It also requires numerous quantitative inputs, as listed in Section 9, which must be obtained for each mission scenario of interest.

The FLCCPSU model is intimately tied to the mission scenario, which includes the maintenance scenario, the dispatch policy, the cost of failing to sortie, the cost of losing a vehicle, and other mission-specific parameters. Thus, while the general modeling approach may have utility for many Army missions, the model presented in Section 9 itself can not be viewed as being a formulation which is valid independent of the application.

11.6. Verification of Weight, Power, and Volume Predictions

Section 9 of this report contains simple models for the total fielded weight, power, and volume (WPV) of an AFTA implementation. These models are parameterized upon the number of LRMs and LRUs in the implementation, their power consumption and volume, and other parameters. WPV-related characteristics of each AFTA component are known via empirical measurement or engineering predictions: NDI components' characteristics can be empirically measured, while the NE characteristics can be predicted based on engineering calculations and past experience. As the Brassboard NE design progresses, these predictions will increase in accuracy. The AFTA WPV are estimated by the simple summation of the WPV of the components comprising a configuration. Verification of this linear composition model is straightforward.

This page intentionally left blank.

12. AFTA Architecture Synthesis

AFTA is characterized by numerous physical and operational parameters which may be adjusted to meet the throughput, reliability, availability, and other requirements of a particular mission. The effects of varying these parameters upon the AFTA availability, reliability, weight, power, volume, and cost (for the iterative aircraft mission) are provided by the analytical formulations in Section 9 of this report. The process of suitably adjusting these parameters in conjunction with use of the AFTA analytical models is denoted "architecture synthesis."

In this Section an architecture synthesis procedure is described which uses the mission requirements described in Section 2 and the analytical formulations developed in Section 9—this procedure is but one of many equally valid procedures suitable for use at a conceptual study level of detail. This section subsequently demonstrates the use of this procedure to determine tentative AFTA configurations for the helicopter TF/TA/NOE/FCS and Ground Vehicle applications, within the limitations of the incomplete requirements data.

12.1. AFTA Architecture Synthesis

The AFTA architecture synthesis procedure consists of adjustment of the AFTA configurable parameters. A list of these parameters is presented below.

12.1.1. Configurable Parameters

Number of VGs: Selection of the number of VGs is based on the throughput requirements of the application. The AFTA may contain from one to forty VGs. The delivered throughput of a VG has a second-order dependence on its redundancy level which is sufficiently minor to be neglected in this study. The exact relationship between throughput overhead and redundancy level will be measured as the AFTA design and development proceeds.

Redundancy level of each VG: Multiple PEs in the AFTA can be formed into redundant synchronous Virtual Groups (VGs) to achieve a degree of tolerance of random hardware faults. Each VG may have a different redundancy level. A VG's redundancy level may be either simplex, triplex, or quadruplex. A simplex VG has little if any fault tolerance, a triplex VG is fail-operational/fail-safe, and a quadruplex VG is fail-operational/fail-operational/fail-safe.

Number of Processing Elements: The number of VGs and the redundancy level of each VG determines the number of Processing Elements required. Each FCR in an AFTA may possess up to eight active Processing Elements at a given time. A minimal AFTA configuration would consist of four FCRs, three of which possess a single PE. A maximal AFTA configuration would consist of five FCRs, each of which possesses eight PEs for a total of forty PEs. Spare PEs may be added to any FCR, but only eight may use that FCR's network element at any given time.

Number of Fault Containment Regions: An AFTA implementation may possess either four or five FCRs, depending on the reliability and throughput requirements of the application. A four-FCR AFTA is fail-operational/fail-safe while a five-FCR AFTA is fail-operational/fail-operational/fail-safe.

Number of Network Elements: Each FCR of an AFTA must possess at least one network element. Additional spare Network Elements can be added to the AFTA.

Number of Power Conditioners: Each FCR must possess at least one power conditioner. Additional spare PCs may be added for exhaustion resilience.

Number of I/O Controllers: There is currently no constraint on the number or types of IOCs resident in any given FCR. Neither is there any current constraint on the number or type of input or output devices that they control.

Redundancy management policy: Wide latitude exists regarding the management of the AFTA's reconfigurable processing resources. Selection of a redundancy management policy is dependent upon the real-time constraints of the VG in which a fault is detected, the mission duration and phase, the resources (throughput, memory, bandwidth, etc.) which can be devoted to the recovery process, and other considerations. In addition, the redundancy level of any VG can be varied at any time in response to faults, changes in mission mode, and testing status.

12.1.2. AFTA Architecture Synthesis Procedure

The simplified AFTA architecture synthesis procedure is as follows. It is assumed that the delivered throughput, availability, and reliability requirements are known.

Step 1. From the throughput requirements and the available throughput per VG, determine the number of VGs required using the delivered throughput model described in

Section 8. The analytical models provide a curve of delivered throughput versus the number of VGs, which facilitates this selection. The number of VGs may also be determined based on other criteria such as functional partitioning or prior experience.

Step 2. From the reliability requirements (either mission reliability or vehicle reliability), the mission characteristics (environment, duration, etc.), and the selected mission redundancy management strategy, determine the (mission or vehicle) redundancy level(s) of the AFTA's VGs using the models described in Section 9. The analytical models provide a curve of reliability versus the number of VGs and VG redundancy level. The VG redundancy levels may also be chosen based on some other criteria, such as fail-op/fail-safe, etc.

Step 3. From the availability requirements, determine how many spare PEs are needed in each FCR using the sortie availability model described in Section 9. The analytical models provide a curve of availability versus the number of VGs, VG redundancy level, number of spare FCRs, and number of spare PEs per FCR. The number of spares may also be chosen based on some other criteria.

Step 4. Using the weight, power, and volume models described in Section 9, determine the WPV of the configuration. The analytical models provide a curve of WPV versus the number of PEs and FCRs.

Step 5. For the iterative aircraft mission, the FLCCPSU model provides an estimate of the life cycle costs associated with the configuration. The analytical models provide curves of FLCCPSU versus the parameters mentioned above and other cost-related inputs.

12.2. AFTA Architecture Synthesis

12.2.1. AFTA Characteristics Common to Both Missions

Certain AFTA characteristics are to some extent independent of the mission. Assuming that both missions utilize similar PEs (e.g. the R3000 PE for the flight vehicle mission and the 68040 for the ground vehicle mission), these characteristics include the delivered throughput.

12.2.1.1. Delivered Throughput

The following chart depicts the delivered throughput of AFTA as a function of the number of PEs and the redundancy level into which the PEs are grouped. It is assumed that PEs having a raw throughput of 20 MIPS (e.g., 68040 or R3000-class) are used, all

VGs in a configuration possess the same redundancy level, and the net overheads due to the AFTA Operating System and FDIR are equal to 20%.

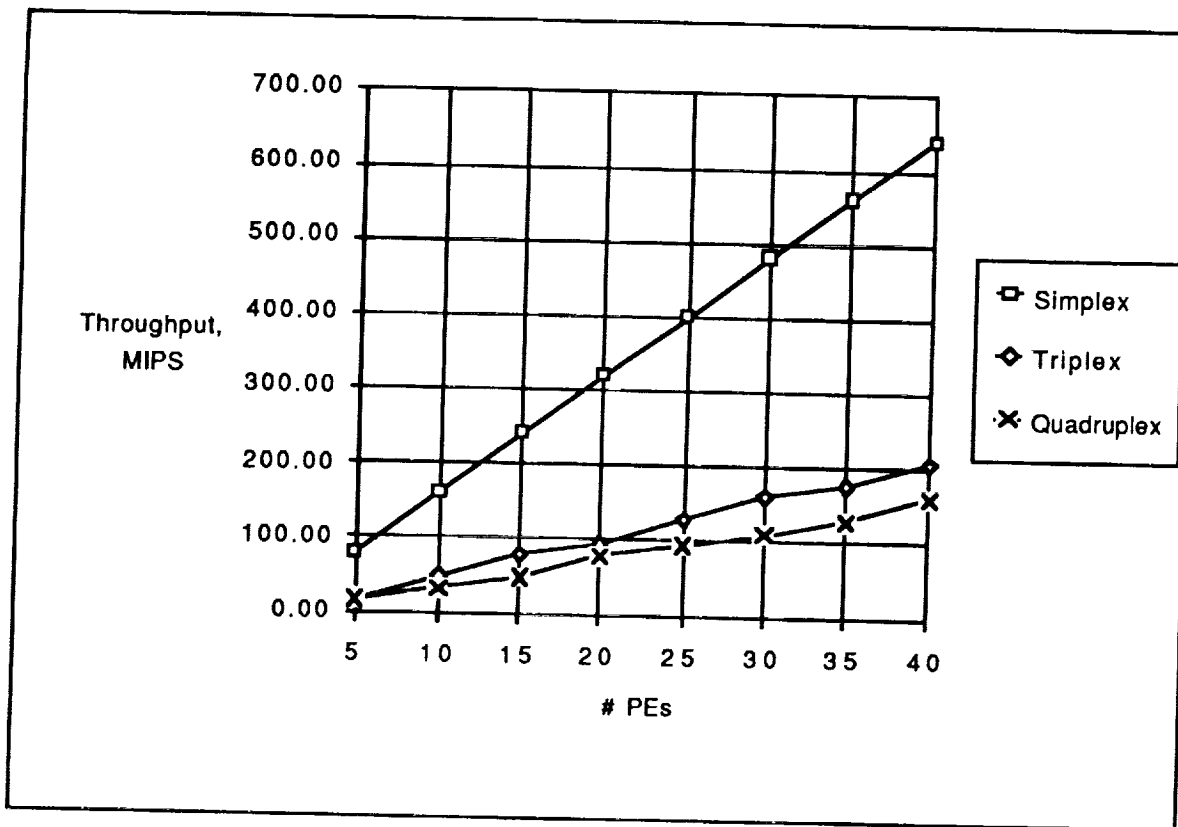


Figure 12-1. AFTA Delivered Throughput vs. Number of Processing Elements

12.2.2. AFTA Configuration for TF/TA/NOE/FCS Mission

The preliminary requirements analysis of Section 2 indicates that nine processing sites are needed for flight-critical processing functions in the helicopter TF/TF/NOE/FCS mission. For reasons outlined in Section 2, it is thought that the throughput obtained using this processor count greatly exceeds that actually needed when operating system overheads are extracted and processor throughputs are increased. Therefore, when the presentation requires for conciseness and concreteness that some number of VGs be specified, it will be assumed that six VGs are needed for flight-critical processing functions. It should be borne in mind that this number may still represent a throughput overkill, and that the AFTA analytical models produce results for any realizable AFTA processor count.

12.2.2.1. Analytical Results

12.2.2.1.1. Failure Rates[†]

The AFTA component failure rates were calculated assuming that the hiatus environment corresponds to the Ground, Fixed (GF) environment, and the mission environment corresponds to the Rotary Wing Aircraft (AR) environment, both described in MIL-HDBK-217E.

Component	GF failure rate, per h	AR failure rate, per h
PE ^{††}	1.92E-5	6.58E-5
NE*	4.08E-5	1.85E-4
PC**	1.59E-5	5.40E-5
FCR Bus ^{†*}	1.92E-6	6.58E-6

Table 12-1. AFTA Component Failure Rates for Helicopter Mission Scenario

12.2.2.1.2. Reliability

For the rotary wing aircraft mission, the AFTA's reliability depends upon the VG redundancy level and the duration of the mission. The following two charts show the AFTA reliability for AFTA configurations composed of all simplex, all triplex, and all quadruplex VGs for mission durations of one and four hours and for AFTA configurations comprising four FCRs (curves labeled with the "-4" suffix) and five FCRs (curves labeled with the "-5" suffix). The baseline NE as described in Section 4 is assumed to be used. I/O failures are not included, and in-flight redundancy management corresponds to the graceful degradation policy described in Sections 5 and 9.

[†] Permanent failure rates only.

^{††} Extrapolated from Lockheed Sanders R3000 VME PE sales literature.

* Baseline NE.

** Extrapolated from Varo Industries JIAWG 27VDC to 5VDC@50A module data.

^{†*} Assumed to be 10% of the PE failure rate.

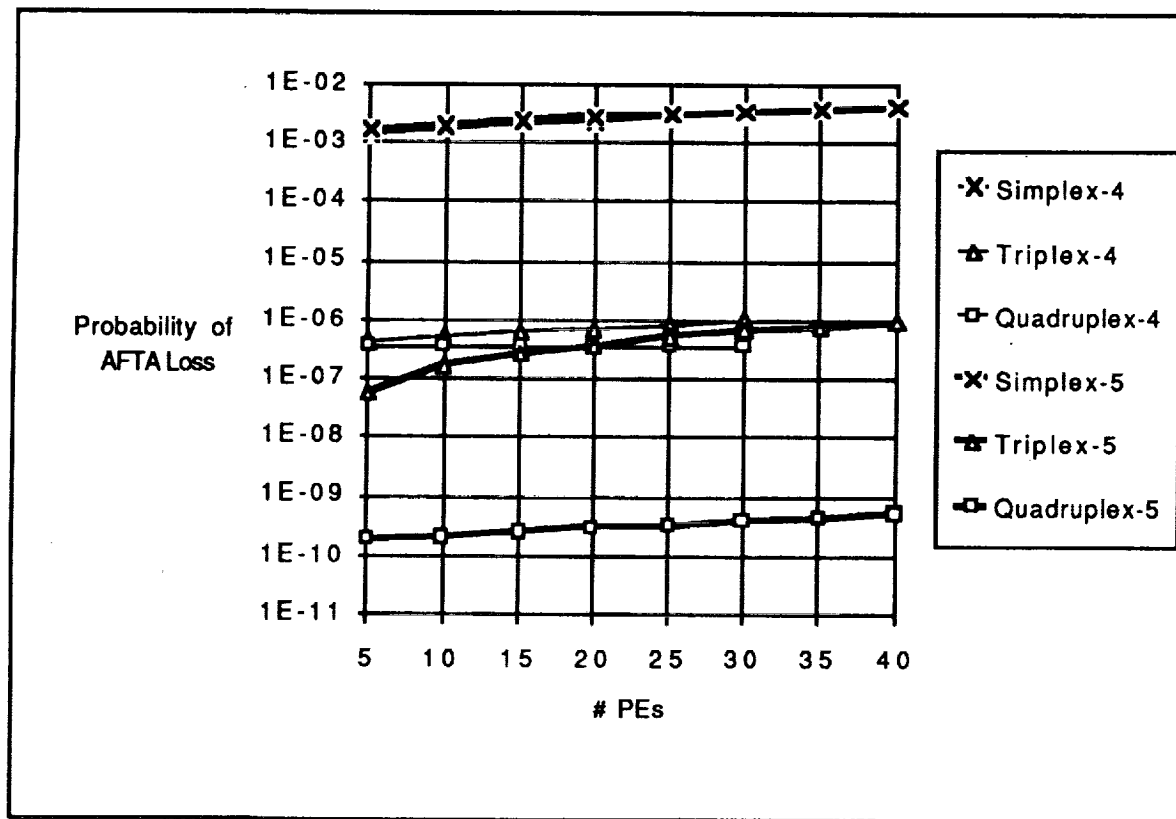


Figure 12-2. Probability of AFTA Failure for 1-hour Rotary Wing Aircraft Mission

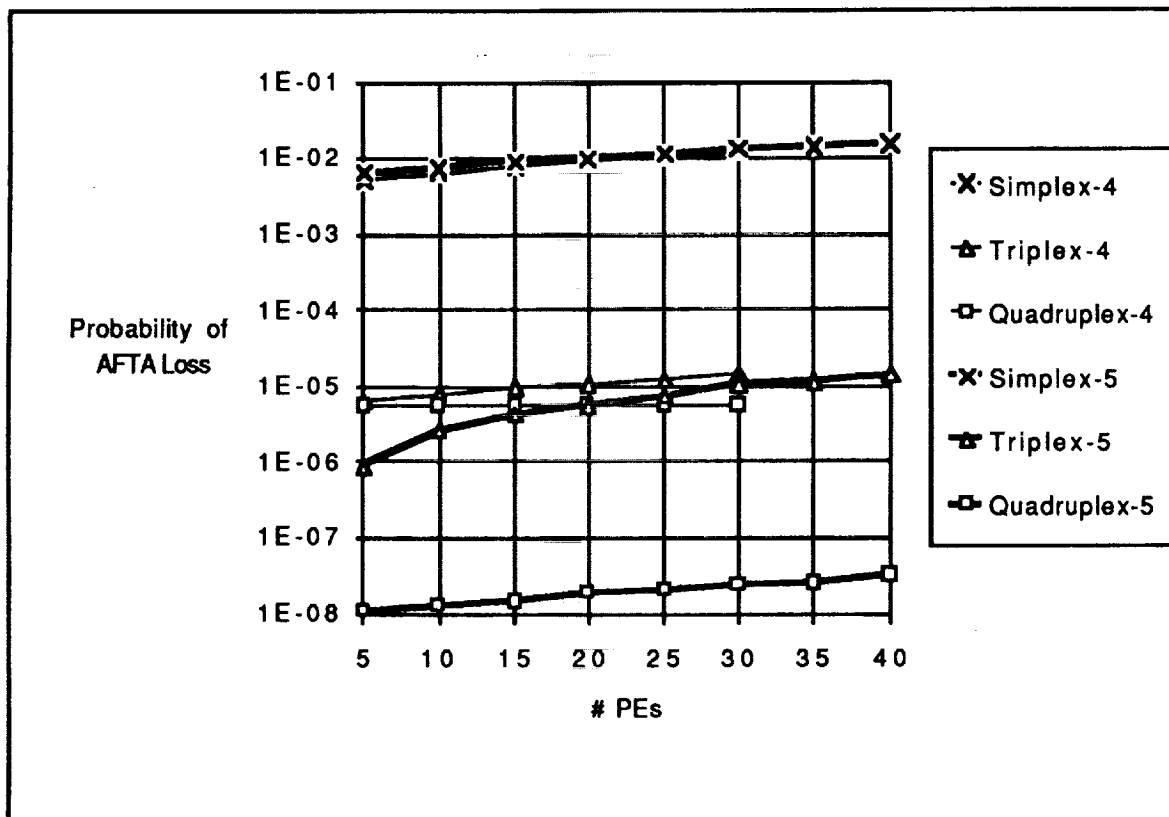


Figure 12-3. Probability of AFTA Failure for 4-hour Rotary Wing Aircraft Mission

As expected, the all-simplex VG configurations have significant higher failure probability than the redundant configurations and can be eliminated from further consideration for flight critical processing on this ground. The all-triplex VG configurations have intermediate reliability levels—their variation with the number of PEs in the configuration illustrates that PE failure, as opposed to NE failure, is the dominant failure mode. Interestingly, the all-quadruplex VG configuration residing in four FCRs has reliability commensurate with the triplex configurations—AFTA failure in this configuration is dominated by NE failure, as indicated by its flat response to the number of PEs in the configuration. This implies that if, for whatever reason, only four FCRs can be supported in the vehicle, then no improvement in mission reliability is achieved by utilizing quadruplex VGs instead of triplex VGs for the mission times of interest. The all-quadruplex VG configuration residing in five FCRs surpasses all configurations in reliability, while its variation of reliability with respect to the number of PEs in the configuration indicates that AFTA failure in this configuration is dominated by PE loss.

The probability of AFTA failure scales quadratically with the mission time for the all-triplex VG configurations and cubically with the all-quadruplex configuration residing in

five FCRs; this scaling is typical of attrition-dominated triplex and quadruplex systems. It is interesting to note, however, that the all-quadruplex configuration residing in four FCRs anomalously scales quadratically with mission time in a manner more reminiscent of a triplex system. This can be explained by the fact that AFTA loss probability for this configuration is dominated by the failure of NEs, as evidenced by its flat response to variation in the number of PEs in the configuration.

12.2.2.1.3. Throughput-Reliability Tradeoff

From the VG versus delivered throughput and reliability versus # PEs curves, a composite chart (Figure 12-4) can be constructed which directly shows the tradeoff between AFTA's delivered throughput and reliability.

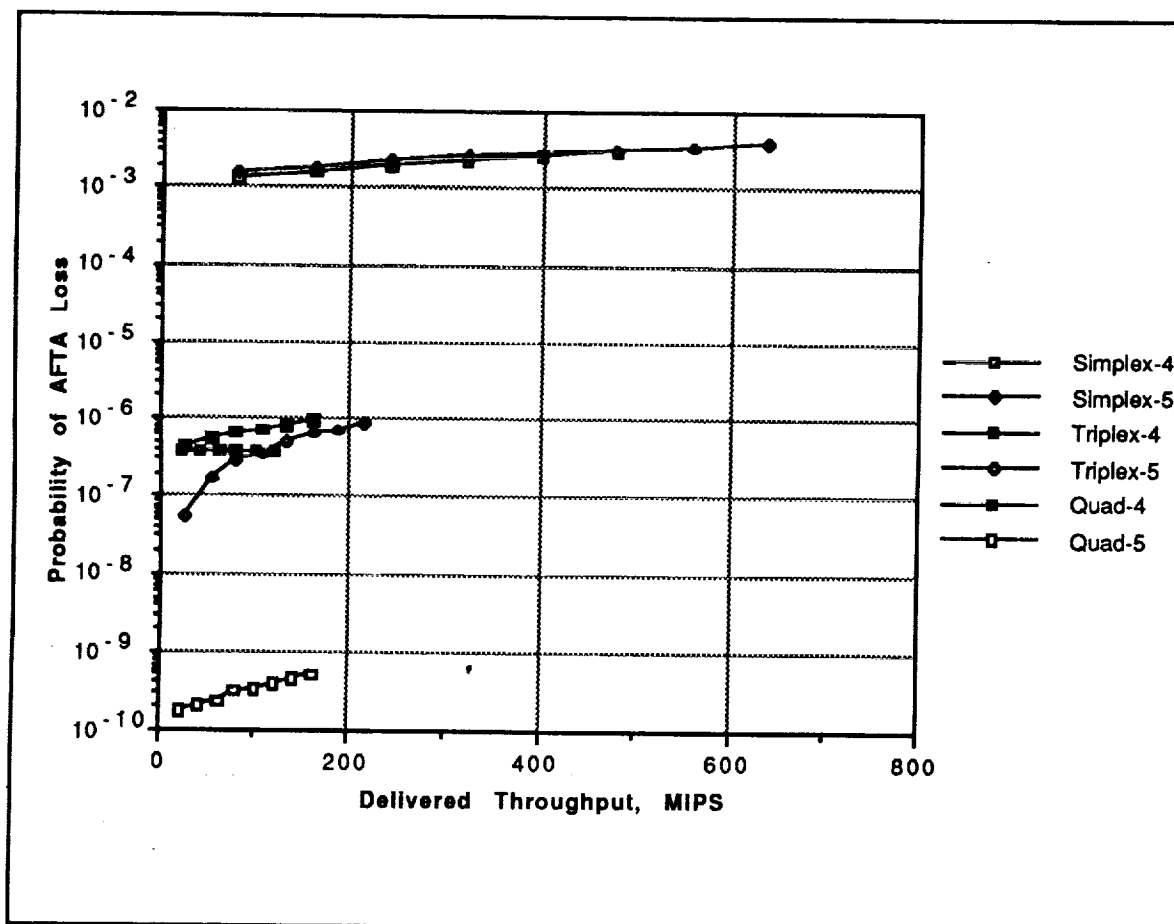


Figure 12-4. Delivered Throughput vs. AFTA Failure Probability for 1-hour Rotary Wing Aircraft Mission

12.2.2.1.4. Effect of VHSIC/VLSI Network Element Technology on Reliability

Based on the Network Element failure rate calculations presented in Section 9, the extensive use of VHSIC/VLSI technology to fabricate the Network Element would increase its MTBF in the rotary wing aircraft environment by a factor of approximately 1.64. The analytical models can be used to translate this component MTBF improvement into mission reliability improvement. The results of the analysis using the VHSIC/VLSI-based "High End" NE are presented in Figure 12-5 for the 1-hour rotary wing aircraft mission. The relative improvement in failure probability is shown in Figure 12-6, which plots the failure probability of an AFTA using the Baseline NE divided by the failure probability of an AFTA using the VHSIC/VLSI-based NE.

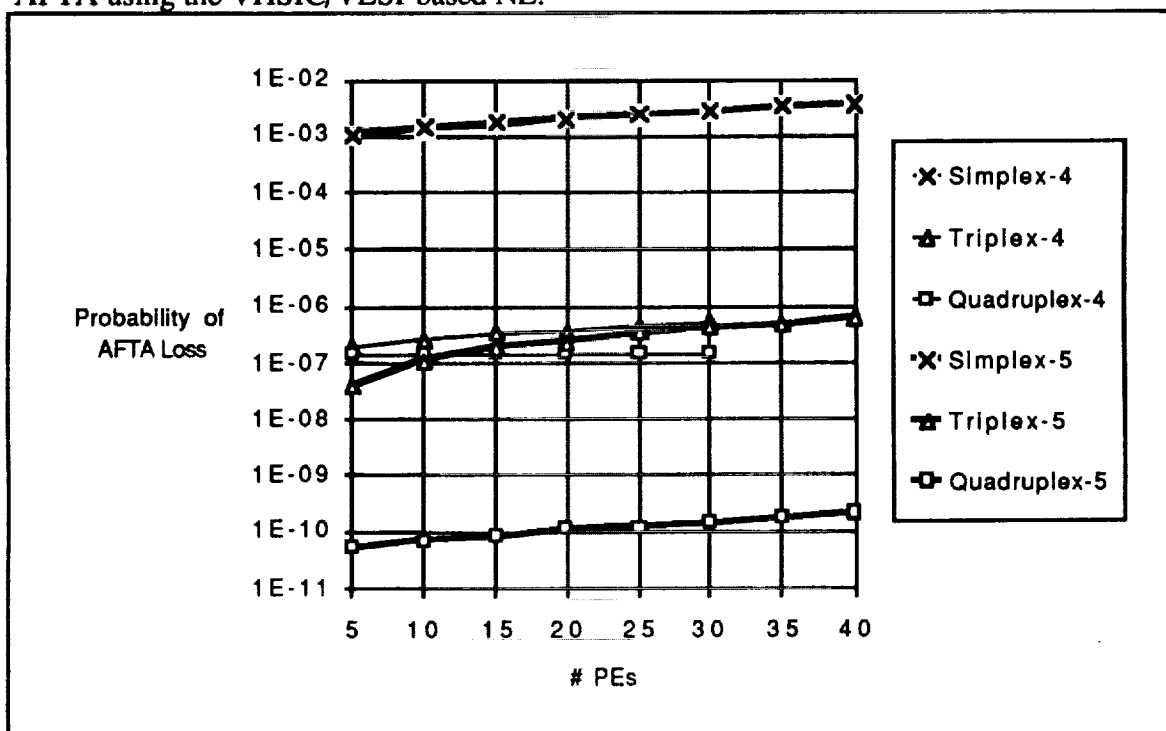


Figure 12-5. Probability of AFTA Failure for 1-hour Rotary Wing Aircraft Mission using VHSIC/VLSI-based Network Element

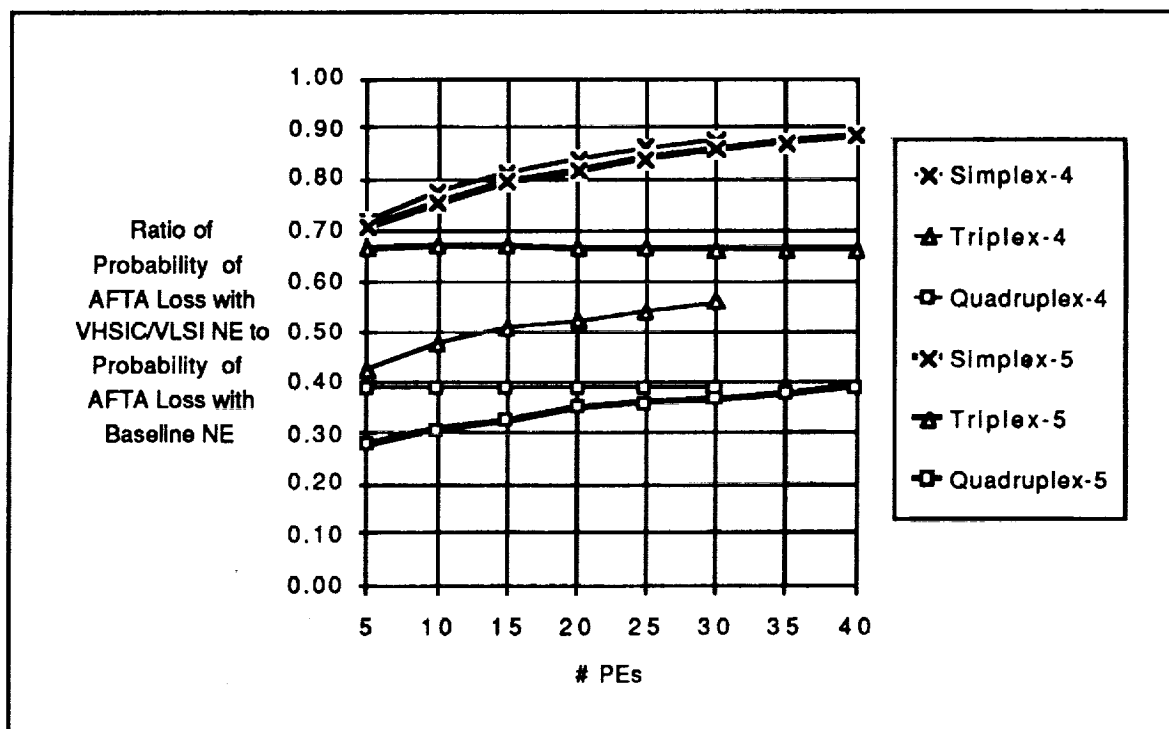


Figure 12-6. Ratio of Probability of AFTA Failure for 1-hour Rotary Wing Aircraft Mission: Baseline NE divided by VHSIC/VLSI-based NE

12.2.2.1.5. Unavailability

From the throughput requirements one may determine the number of VGs required to perform the mission's functions. From the reliability models one may determine the minimum redundancy level these VGs must possess at sortie to meet the mission's reliability requirements. This complement of resources is denoted the Minimum Dispatch Complement (MDC). If MDC is not available at sortie due to faults, then the vehicle can not sortie. AFTA allows the addition of spare components to attempt to increase mission availability.

Figure 12-7 shows the effect on mission availability of adding spare PEs in each FCR and spare FCRs to an AFTA, assuming that an MDC of six VGs and four FCRs are needed to sortie. The analysis was performed for configurations comprising all simplex, all triplex, and all quadruplex VGs. The hiatus interval is assumed to be 23 hours at Ground, Fixed failure rates. The curves labeled with a "4" suffix refer to a configuration containing no spare FCR, while the curves labeled by the "5" suffix refer to configurations containing a spare FCR. For the given model input parameters, addition of more than a single spare PE per FCR does not significantly enhance availability. Addition of a spare FCR increases availability somewhat, while the combination of one spare PE per FCR and one spare FCR significantly enhances availability.

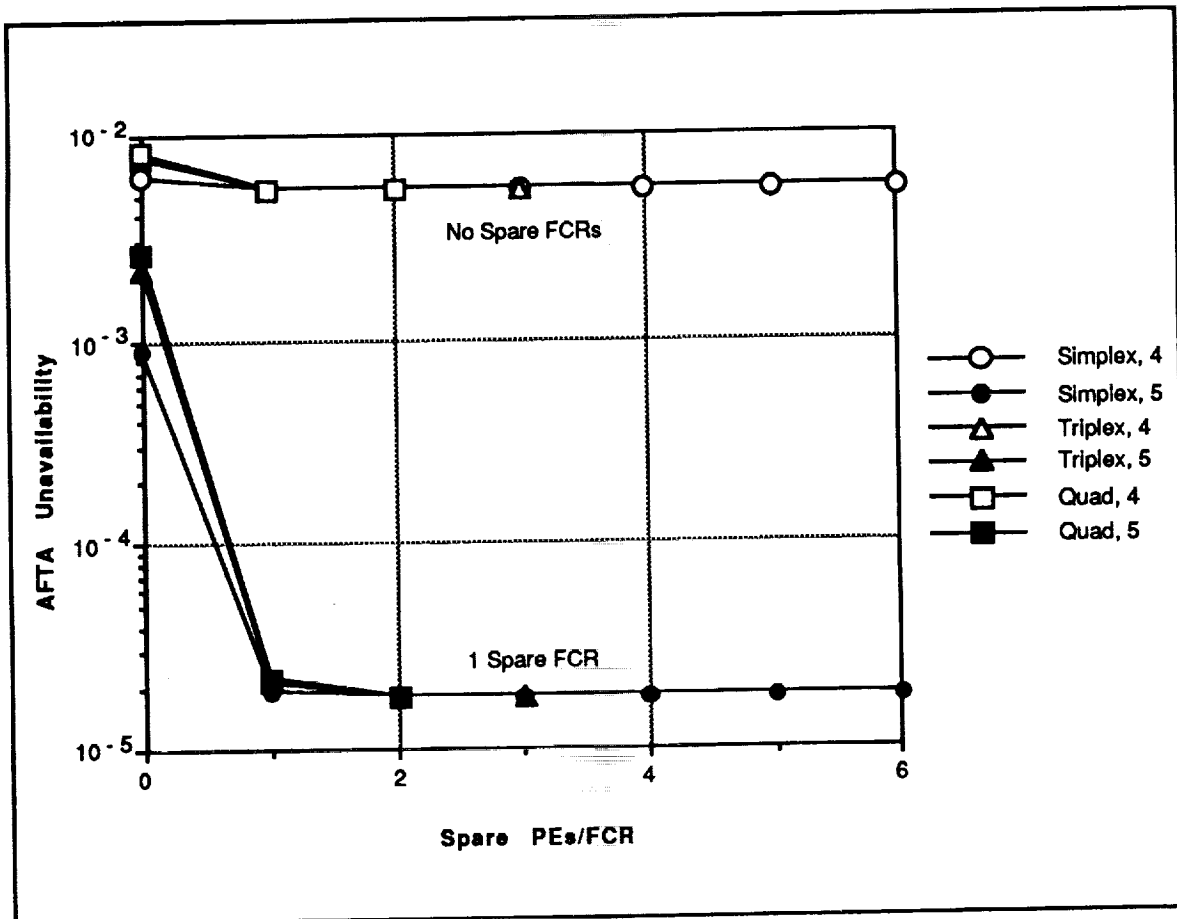


Figure 12-7. AFTA Unavailability after 23-hour Hiatus for Six-VG/Four-FCR MDC

Figure 12-8 shows the effect on mission availability of adding spare PEs in each FCR, assuming that six VGs and five FCRs are needed to sortie. The analysis was performed for configurations comprising all simplex, all triplex, and all quadruplex VGs and the hiatus interval is assumed to be 23 hours at Ground, Fixed failure rates. Again, addition of more than a single spare PE per FCR does not significantly enhance availability. Note that a spare FCR can not be added to this MDC configuration since the number of FCRs would then exceed that supported by AFTA. Consequently the extremely low unavailability levels associated with the combined spare PEs and FCR can not be achieved.

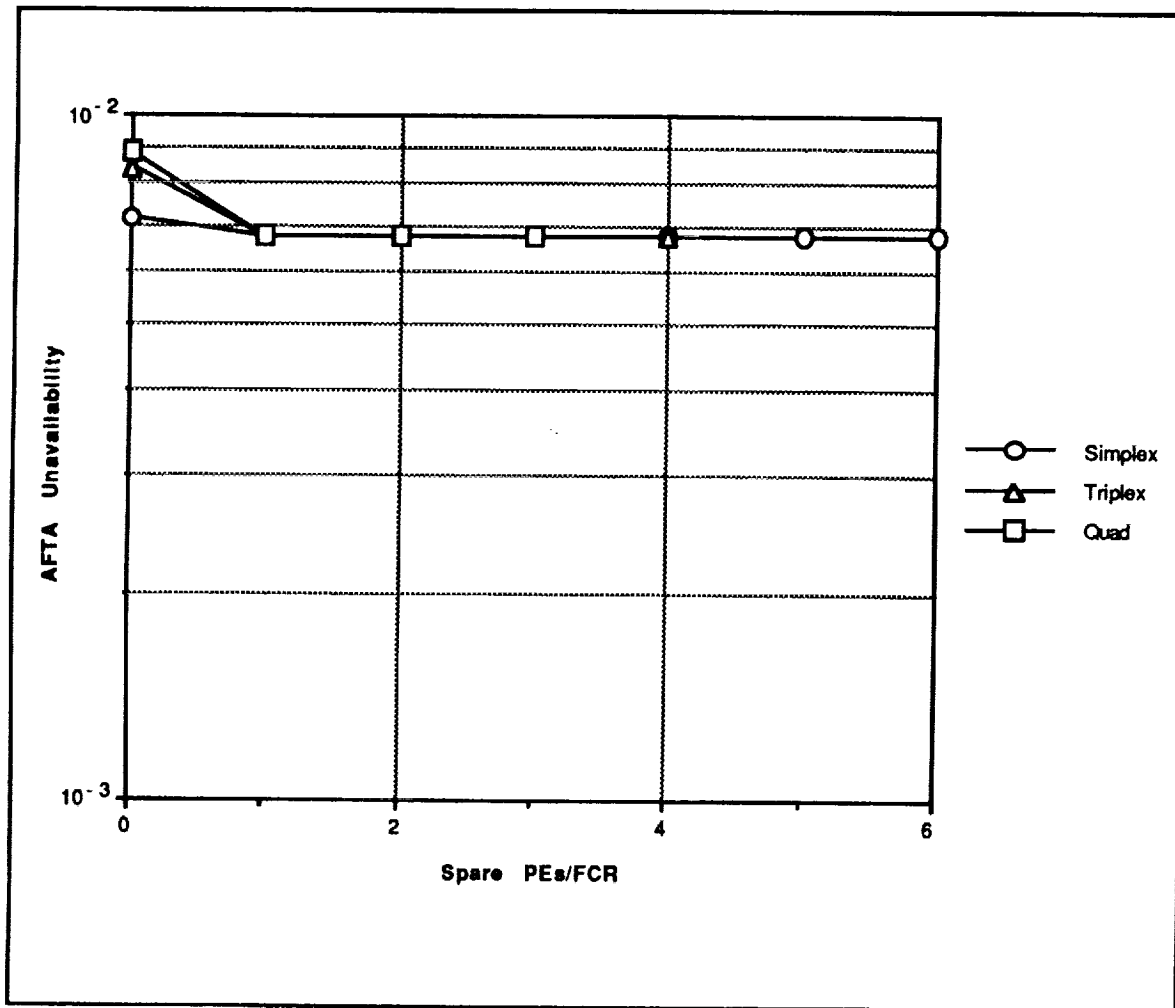


Figure 12-8. AFTA Unavailability after 23-hour Hiatus for Six-VG/Five-FCR MDC

12.2.2.1.6. Weight

It is assumed for the calculation of the AFTA physical parameters for the helicopter mission that JIAWG-class SEM-E packaging is used. Representative component weights are enumerated in Table 12-2.

PE	1 lb.
NE†	1.5 lb.
Rack	5 lb.
PC	3 lb.

Table 12-2. AFTA Component Weights for Helicopter AFTA

The weight of AFTA for the helicopter mission is depicted below as a function of the number of PEs and FCRs.

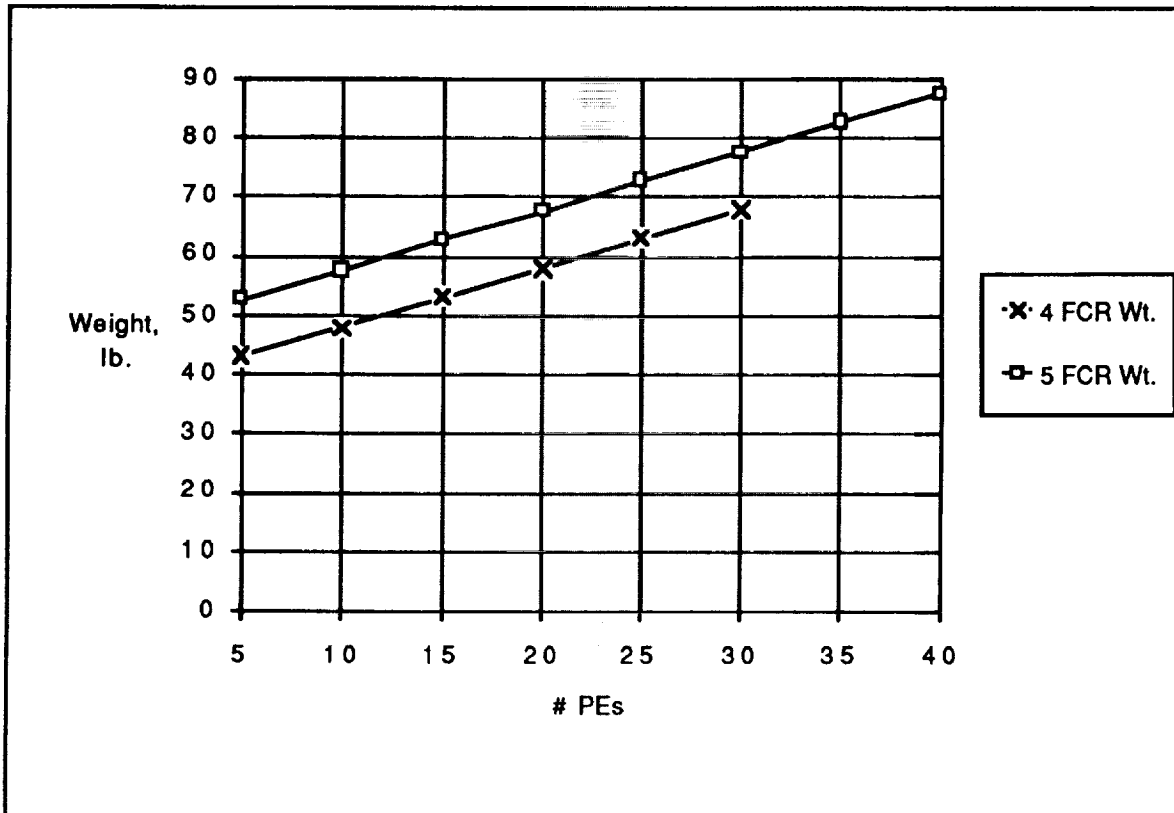


Figure 12-9. AFTA Weight for Helicopter Mission

12.2.2.1.7. Power

Representative power consumptions of JIAWG-class components are enumerated in Table 12-3.

† Includes optical splitters.

PE Power	20 W
NE Power	20 W
Bus Power	1 W
PC Efficiency	90%

Table 12-3. AFTA Component Powers for Helicopter AFTA

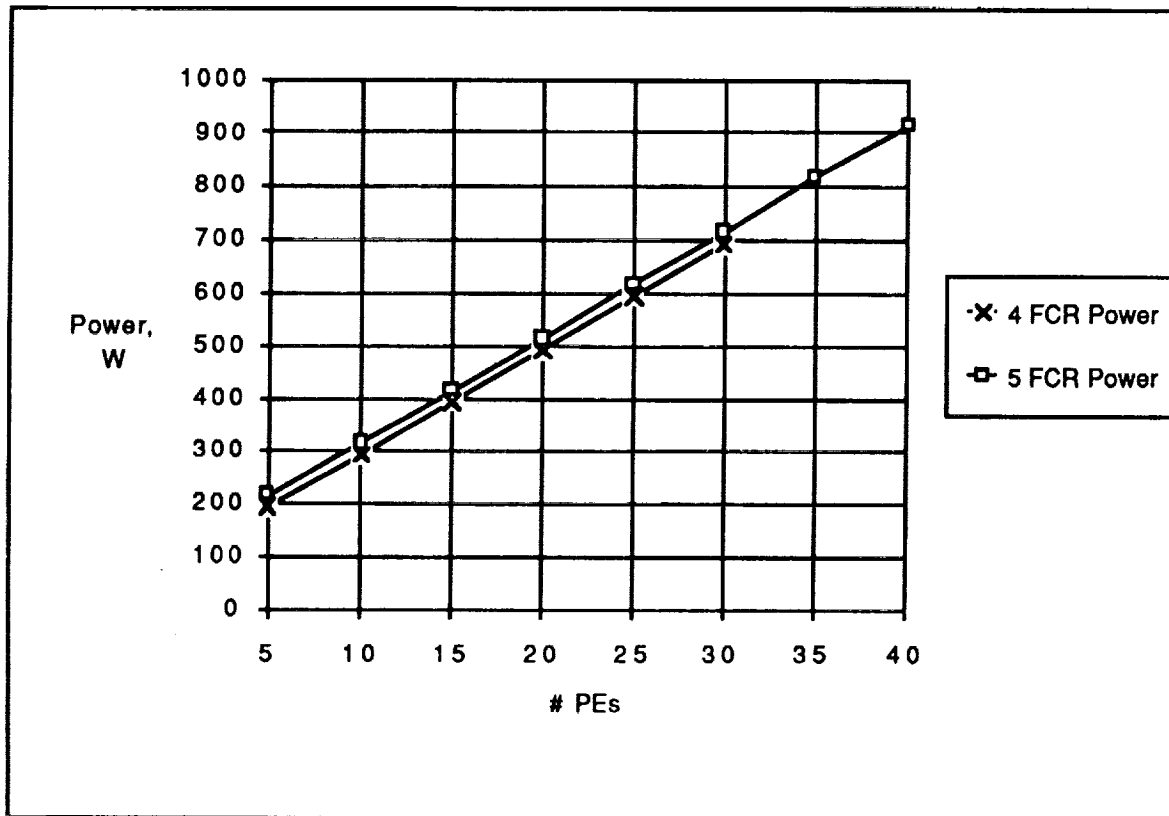


Figure 12-10. AFTA Power for Helicopter Mission

12.2.2.1.8. Volume

The volumes of JIAWG-class SEM-E AFTA modules are enumerated in Table 12-4.

PE Slots	1
NE Slots	1
PC slots	2
Rack Ends (# slots)	2
Slot Volume	1.77E-02 cu. ft. (6.88 x 7.42 x 0.60 in ³)

Table 12-4. AFTA Component Volumes for Helicopter AFTA

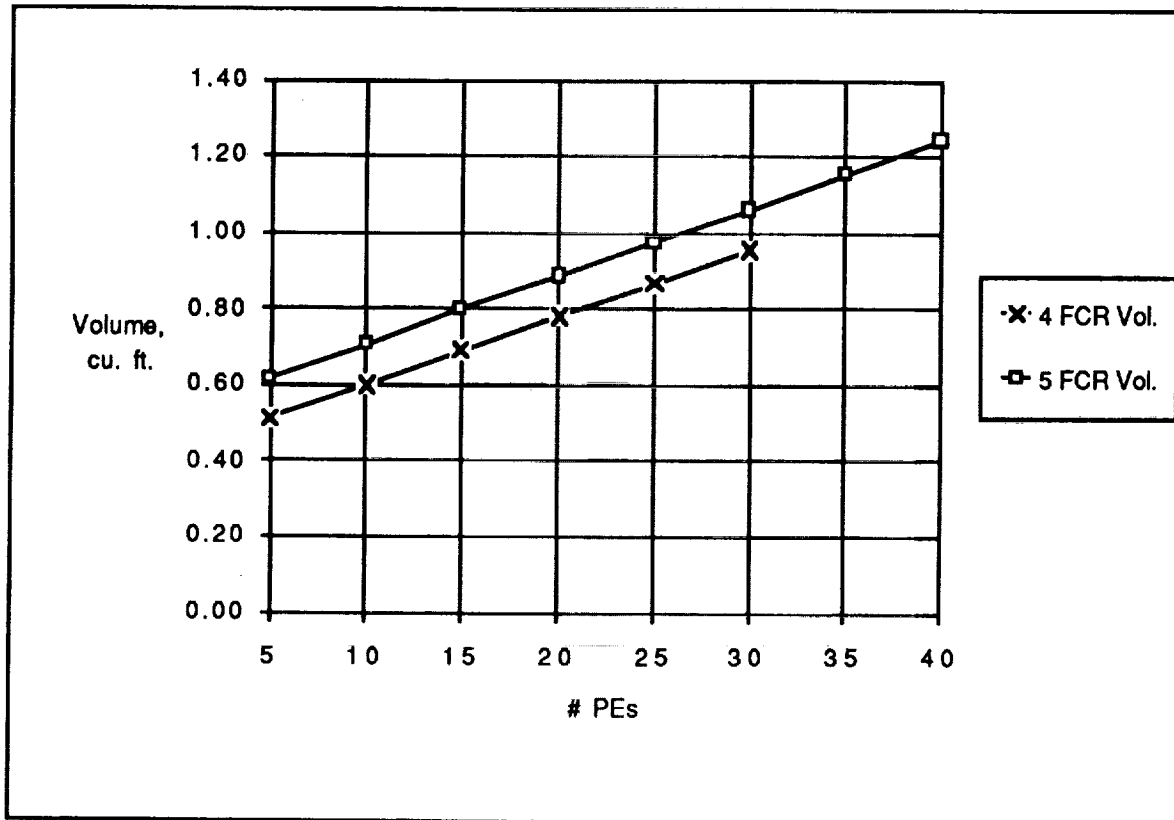


Figure 12-11. AFTA Volume for Helicopter Mission

12.2.2.1.9. Cost

The Fleet Life Cycle Cost per Service Unit model described in Section 9 generates voluminous cost data for many possible configurations of AFTA. For conciseness, this report only presents the cost data for the one-hour Rotary Wing Aircraft mission for an MDC of six VGs. The input parameter values for the cost model are listed below. These parameters are for illustrative purposes only, and should be updated as additional

information becomes available during AFTA development and mission requirements acquisition.

Parameter	Value
Fleet service life	100,000 hours
Hiatus time	23 hours
Sortie duration	1 hour
Number of vehicles required per sortie	100
Baseline vehicle cost	\$6,000,000
AFTA LRM cost	\$10,000
Number of LRMs in AFTA	Depends on Configuration
Cost of AFTA rack	\$10,000
Number of racks in AFTA	Depends on Configuration
Manpower cost for field repair, per hour	\$200
Mean time for field repair, hours	4
Manpower cost for depot diagnosis, per hour	\$100
Mean time for depot diagnosis, hours	4
Manpower cost for depot repair, per hour	\$200
Mean time for depot repair, hours	4
Depot condemnation ratio	1
LRM/LRU refurbishment parts cost	\$1,000

Table 12-5. FLCCPSU Input Parameters for Rotary Wing Aircraft Mission

For clarity of presentation, the cost analysis results are presented separately for three different AFTA configurations: a four-FCR MDC configuration with no spare FCR, a four-FCR MDC configuration with one spare FCR, and a five-FCR MDC configuration with no spare FCR. The results are presented in tabular format. The leftmost column indicates the redundancy level of the AFTA's VGs—this may be simplex, triplex, or quadruplex. It is assumed that all VGs in a configuration are of identical redundancy level, an assumption which may be relaxed without difficulty. In addition, for a given VG redundancy level, the leftmost column lists the number of spare PEs per FCR which may be added to increase availability. For conciseness, this parameter is varied from zero to one spare PE per FCR, based on the availability analysis' illustration of the rapidly diminishing return of additional spare PEs for a 23-hour hiatus. The row corresponding to VG

redundancy level and number of spare PEs per FCR contains the corresponding cost data. The column denoted Fleet Life Cycle Cost is the sum of the the cost of unreliability, the cost of maintenance labor, the cost of AFTA spares, the cost of vehicles, and the initial procurement cost of the fleet's AFTAs. In addition, the last column of the tables below indicates the cost of unavailability, i.e., the cost of additional vehicle procurements required to meet the sortie requirement given non-unity AFTA availability. This cost is included in the cost of vehicles and cost of AFTAs columns and is included as a separate italicized column solely for edification.

The results are presented in the following table for an AFTA configuration consisting of a four-FCR MDC and no spare FCR.

VG Redundancy Level	Fleet Life Cycle Cost	Cost of Unreliability	Cost of Maintenance Labor	Cost of Spares	Cost of Vehicles	Cost of AFTAs	<i>Cost of Unavailability</i>
Simplex							
0 spare PEs/FCR	4.03E+09	3.36E+09	5.11E+06	4.26E+07	6.04E+08	1.61E+07	<i>3.88E+06</i>
1 Spare PE/FCR	4.07E+09	3.39E+09	6.13E+06	5.11E+07	6.03E+08	2.01E+07	<i>3.35E+06</i>
Triplex							
0 spare PEs/FCR	7.11E+08	1.81E+06	8.16E+06	6.80E+07	6.05E+08	2.82E+07	<i>4.79E+06</i>
1 Spare PE/FCR	7.23E+08	1.82E+06	9.17E+06	7.64E+07	6.03E+08	3.22E+07	<i>3.42E+06</i>
Quadruplex							
0 spare PEs/FCR	7.24E+08	9.52E+05	9.17E+06	7.64E+07	6.05E+08	3.23E+07	<i>5.10E+06</i>
1 Spare PE/FCR	7.36E+08	9.58E+05	1.02E+07	8.49E+07	6.03E+08	3.62E+07	<i>3.44E+06</i>

Table 12-6. FLCCPSU Output Parameters for Rotary Wing Aircraft Mission, Four-FCR MDC with no spare FCR

The analysis indicates that the minimum cost configuration for a six VG, four-FCR MDC AFTA with no spare FCR consists of six triplex VGs with no spare PEs. The total cost for this configuration is \$711M. Table 12-7 shows the relative contributions to this cost.

Cost of Unreliability	Cost of Maintenance Labor	Cost of Spares	Cost of Vehicles	Cost of AFTAs	Cost of Unavailability
0.26%	1.15%	9.61%	85.48%	3.99%	0.68%

Table 12-7. FLCCPSU Constituent Costs for Four-FCR MDC with no spare FCR

The analytical results are presented in the following table for an AFTA configuration consisting of a four-FCR MDC and one spare FCR.

VG Redundancy Level	Fleet Life Cycle Cost	Cost of Unreliability	Cost of Maintenance Labor	Cost of Spares	Cost of Vehicles	Cost of AFTAs	Cost of Unavailability
Simplex							
0 spare PEs/FCR	4.03E+09	3.36E+09	5.11E+06	4.26E+07	6.01E+08	1.60E+07	5.56E+05
1 Spare PE/FCR	4.06E+09	3.39E+09	6.13E+06	5.11E+07	6.00E+08	2.00E+07	1.16E+04
Triplex							
0 spare PEs/FCR	7.07E+08	1.81E+06	8.16E+06	6.80E+07	6.01E+08	2.81E+07	1.40E+06
1 Spare PE/FCR	7.19E+08	1.82E+06	9.17E+06	7.64E+07	6.00E+08	3.20E+07	1.33E+04
Quadruplex							
0 spare PEs/FCR	7.20E+08	9.52E+05	9.17E+06	7.64E+07	6.02E+08	3.21E+07	1.69E+06
1 Spare PE/FCR	7.32E+08	9.58E+05	1.02E+07	8.49E+07	6.00E+08	3.60E+07	1.41E+04

Table 12-8. FLCCPSU Output Parameters for Rotary Wing Aircraft Mission—Four-FCR MDC with one spare FCR

The minimum cost configuration for a six VG, four-FCR MDC AFTA with one spare FCR again consists of six triplex VGs with no spare PEs. The total cost for this configuration is \$707M. Table 12-9 shows the relative contributions to this cost. Note the reduced cost due to unavailability due to the added FCR.

Cost of Unreliability	Cost of Maintenance Labor	Cost of Spares	Cost of Vehicles	Cost of AFTAs	Cost of Unavailability
0.26%	1.15%	9.61%	85.01%	3.97%	0.20%

Table 12-9. FLCCPSU Constituent Costs for Four-FCR MDC with one spare FCR

Finally, the results are presented below for an AFTA configuration consisting of a five-FCR MDC.

VG Redundancy Level	Fleet Life Cycle Cost	Cost of Unreliability	Cost of Maintenance Labor	Cost of Spares	Cost of Vehicles	Cost of AFTAs	Cost of Unavailability
Simplex							
1 Spare PE/FCR	4.70E+09	4.02E+09	6.39E+06	5.33E+07	6.04E+08	2.01E+07	4.19E+06
Triplex							
0 spare PEs/FCR	7.20E+08	9.08E+05	8.93E+06	7.44E+07	6.05E+08	3.03E+07	5.38E+06
1 Spare PE/FCR	7.35E+08	9.15E+05	1.02E+07	8.50E+07	6.04E+08	3.52E+07	4.30E+06
Quadruplex							
0 spare PEs/FCR	7.36E+08	8.81E+02	1.02E+07	8.50E+07	6.05E+08	3.53E+07	5.71E+06
1 Spare PE/FCR	7.51E+08	8.88E+02	1.15E+07	9.56E+07	6.04E+08	4.03E+07	4.33E+06

Table 12-10. FLCCPSU Output Parameters for Rotary Wing Aircraft Mission-Five-FCR MDC

Once again, the minimum cost configuration for a six VG, five-FCR MDC AFTA consists of six triplex VGs with no spare PEs. The total cost for this configuration is \$720M. Table 12-11 shows the relative contributions to this cost.

Cost of Unreliability	Cost of Maintenance Labor	Cost of Spares	Cost of Vehicles	Cost of AFTAs	Cost of Unavailability
0.13%	1.26%	10.52%	85.55%	4.28%	0.76%

Table 12-11. Minimum-FLCCPSU Constituent Costs for Four-FCR MDC with one spare FCR

Of all the configurations modeled, the lowest-cost configuration consists of a four-FCR MDC AFTA with one spare FCR and no spare PEs at all. Note that this configuration differs from one selected based on maximum reliability (five-FCR MDC containing all quadruplex VGs) or maximum availability (one spare FCR and one spare PE per FCR). It should also be emphasized that these results are presented primarily to illustrate the use of cost modeling to assist in the architecture synthesis process. The actual results obtained

depend strongly on the input parameters and will lead to different conclusions as these parameters change.

12.2.3. AFTA Analysis for Ground Vehicle Mission

12.2.3.1. Throughput

The AFTA throughput for the Ground Vehicle mission is presented in Section 11.2.1.1.

12.2.3.2. Reliability

In the context of the Ground Vehicle mission the AFTA's unreliability contributes to the failure of the vehicle to perform its mission, in accordance with the Ground Vehicle mission state diagram presented in Section 2. This is not assumed to result in loss of the vehicle. Moreover, the relaxed temporal constraints associated with the ground mission and the long mission times dictate the use of an availability maximization redundancy management policy, described as the "processor replacement" option in Section 5 and modeled as ppr in Section 9. We therefore parameterize mission success as the probability that the Minimum Mission Complement (MMC) of VGs and FCRs are available. It is assumed that, regardless of the redundancy level of the VGs at the beginning of the mission, the AFTA can perform its intended function as long as there are at least MMC functioning simplex VGs which may have started out as simplexes, or which may be degraded triplexes or quadruplexes. Therefore the following charts show ppr as the probability that MMC simplex VGs are functional, as a function of MMC and the number of spare PEs per FCR and the number of spare FCRs.

The failure rates are calculated assuming that the mission environment corresponds to the Ground, Mobile (GM) environment as described in MIL-HDBK-217E.

Component	GM failure rate, per h
PE	3.23E-5
NE	9.26E-5
PC	2.67E-5
FCR Bus	3.23E-6

Table 12-12. AFTA Component Failure Rates for Ground Vehicle Mission Scenario

The following charts show the probability that MMC VGs can not be formed from the simplex processing resources in AFTA as a function of the MMC, the number of spare PEs per FCR, and the number of spare FCRs. The curves are presented for four mission times: 8, 24, 168, and 720 hours.

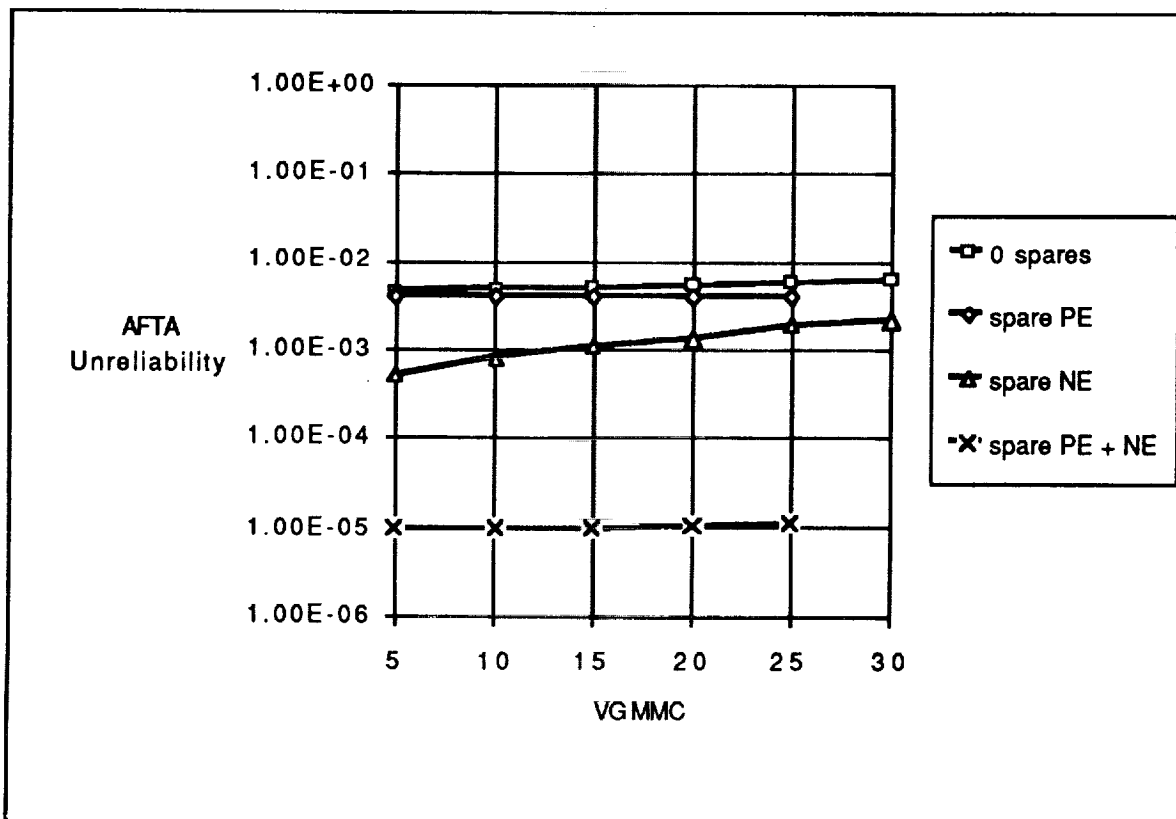


Figure 12-12. AFTA Unreliability for Eight-Hour Ground Vehicle Mission

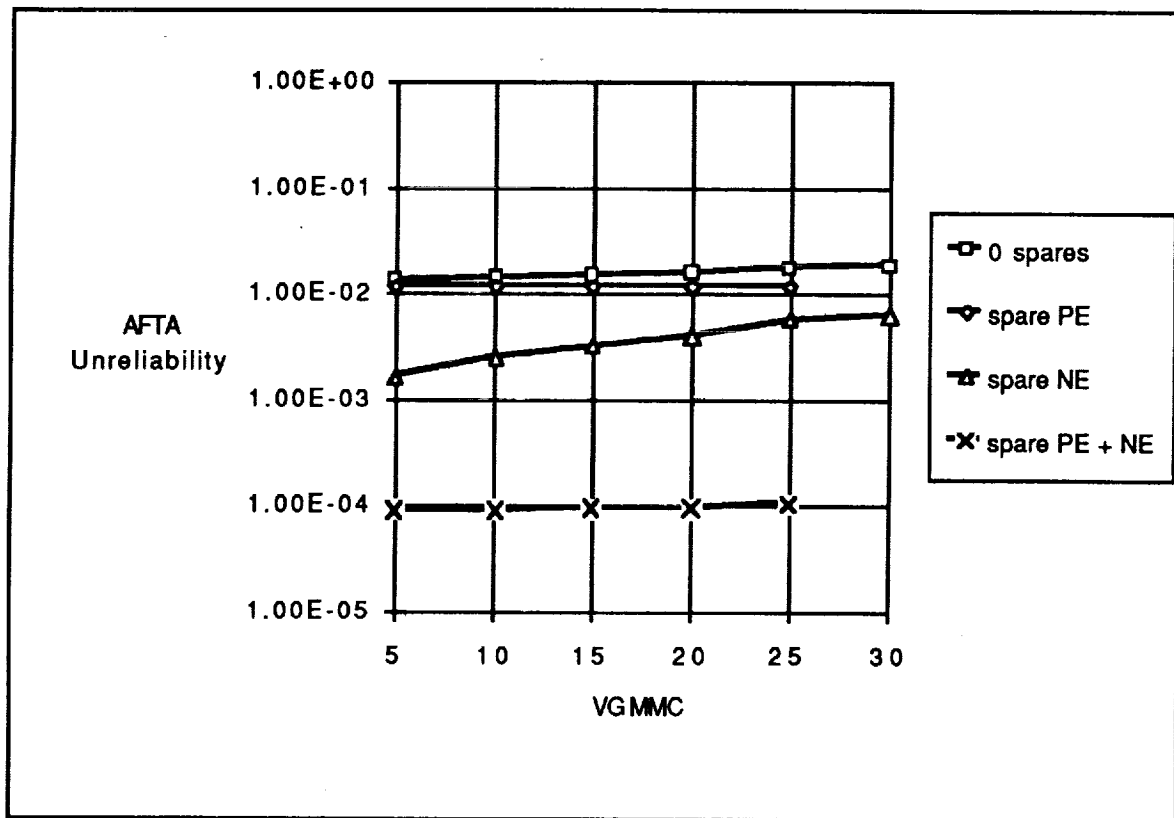


Figure 12-13. AFTA Unreliability for 24-Hour Ground Vehicle Mission

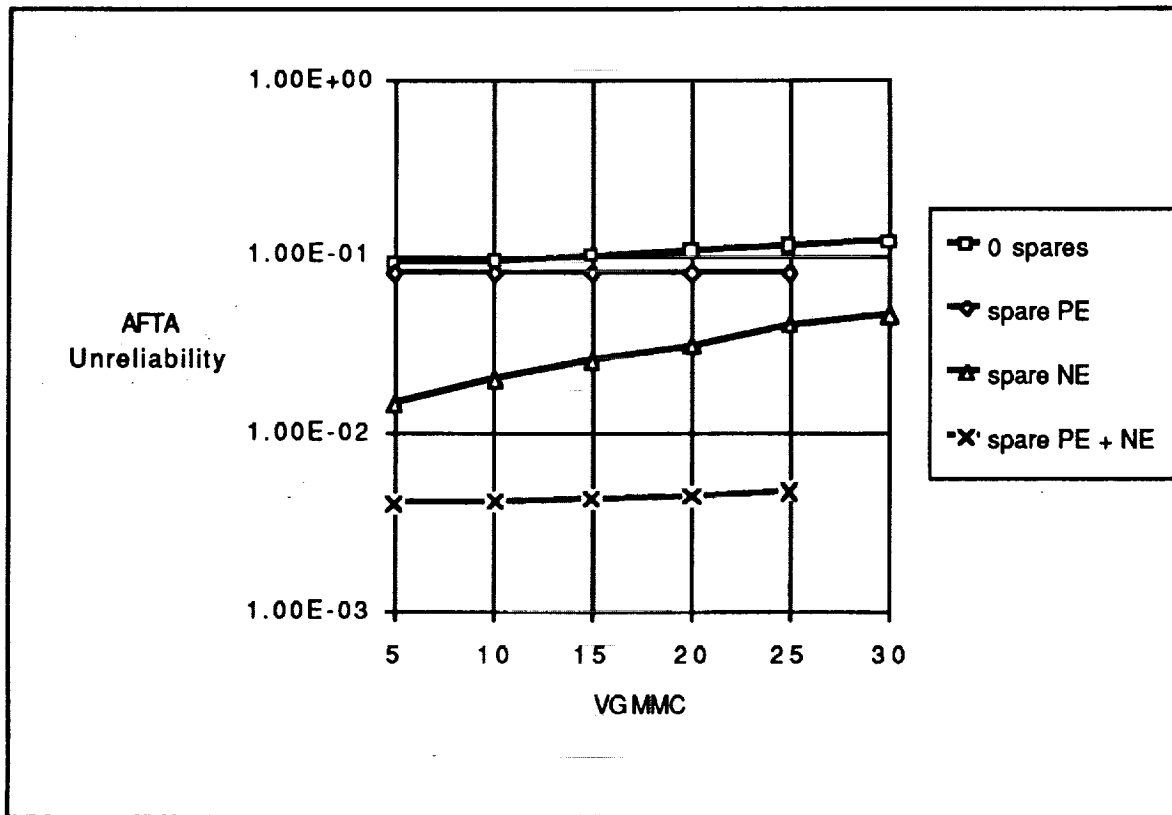


Figure 12-14. AFTA Unreliability for 168-Hour Ground Vehicle Mission

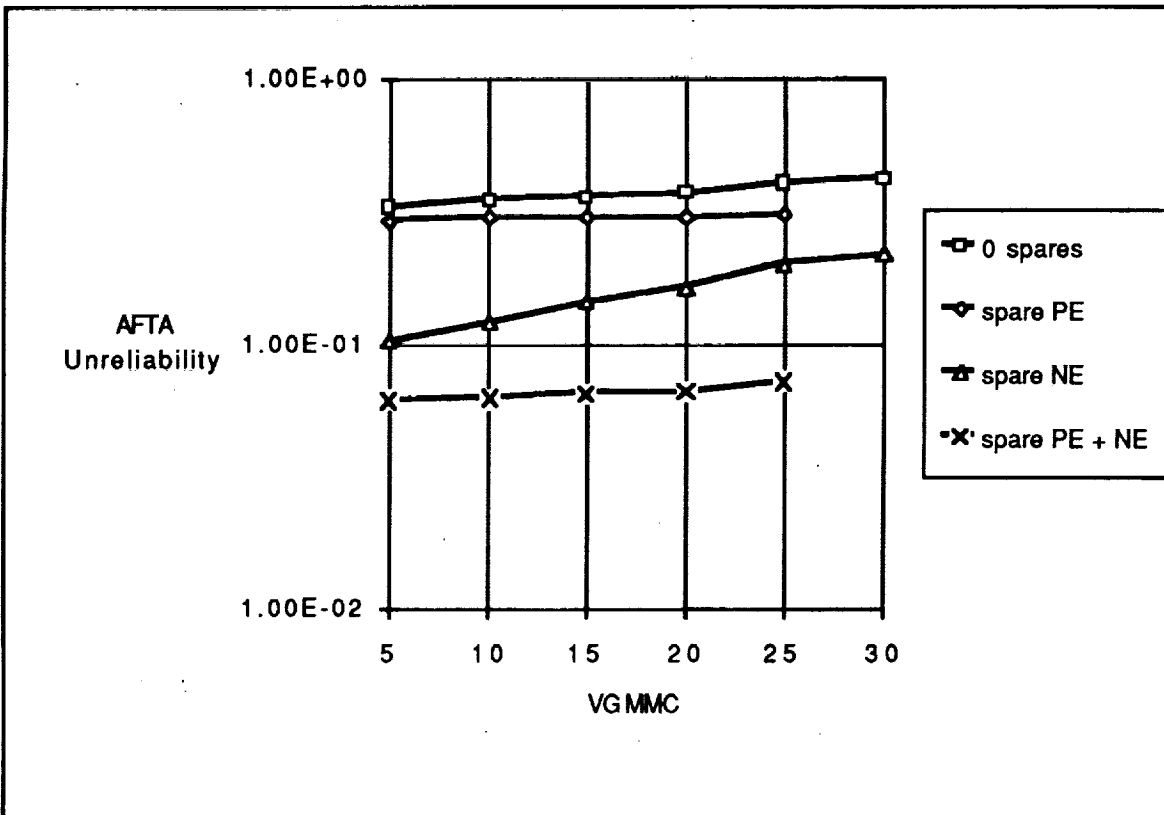


Figure 12-15. AFTA Unreliability for 720-Hour Ground Vehicle Mission

12.2.3.3. Weight

It is assumed for the calculation of the AFTA physical parameters for the Ground Vehicle mission that SAVA-compatible packaging is used. The component weights are enumerated in Table 12-13 (for lack of any better information, these weights are the same as for the SEM-E version of AFTA).

PE	1 lb.
NE [†]	1.5 lb.
Rack	5 lb.
PC	3 lb.

Table 12-13. AFTA Component Weights for Helicopter AFTA

The weight of AFTA for the Ground Vehicle mission is depicted below as a function of the number of PEs and FCRs.

[†] Includes optical splitters.

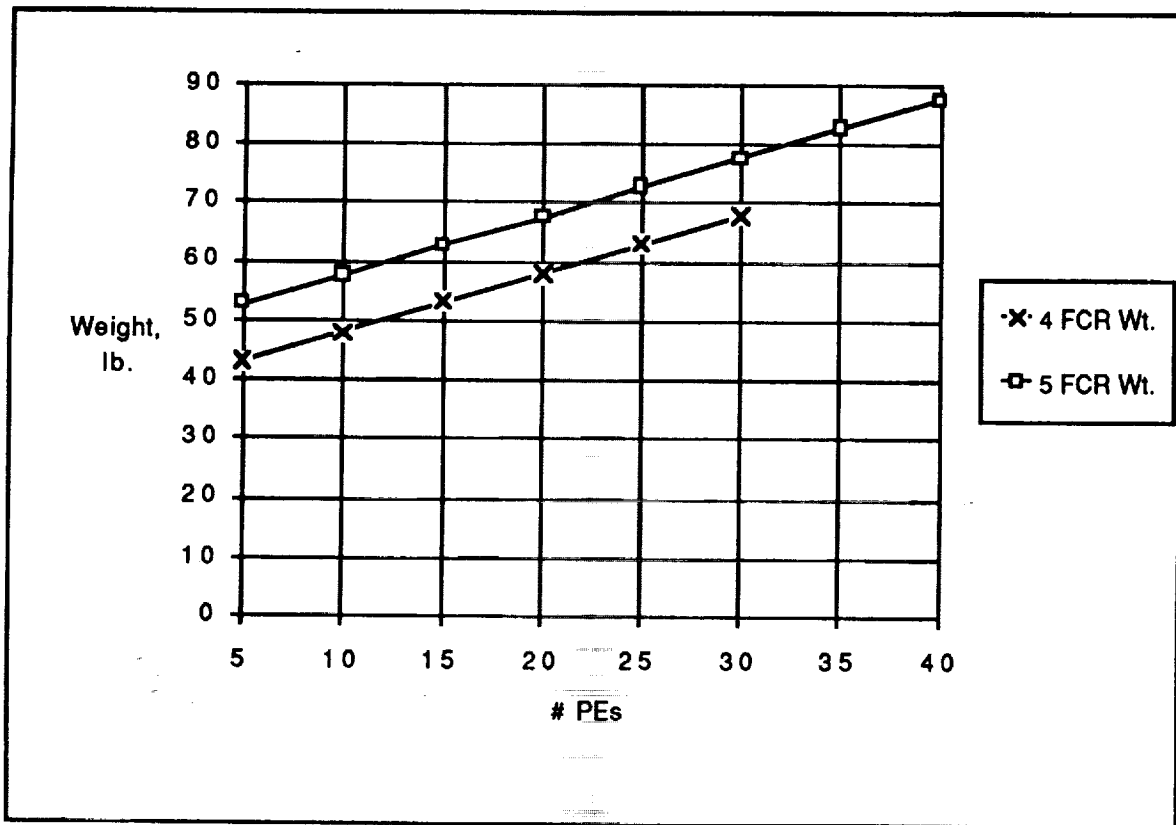


Figure 12-16. AFTA Weight for Ground Vehicle Mission

12.2.3.4. Power

Representative component power consumptions for MIL-STD-344 modules are enumerated in Table 12-14.

PE Power	15 W
NE Power	15 W
Bus Power	1 W
PC Efficiency	90%

Table 12-14. AFTA Component Powers for Ground Vehicle AFTA

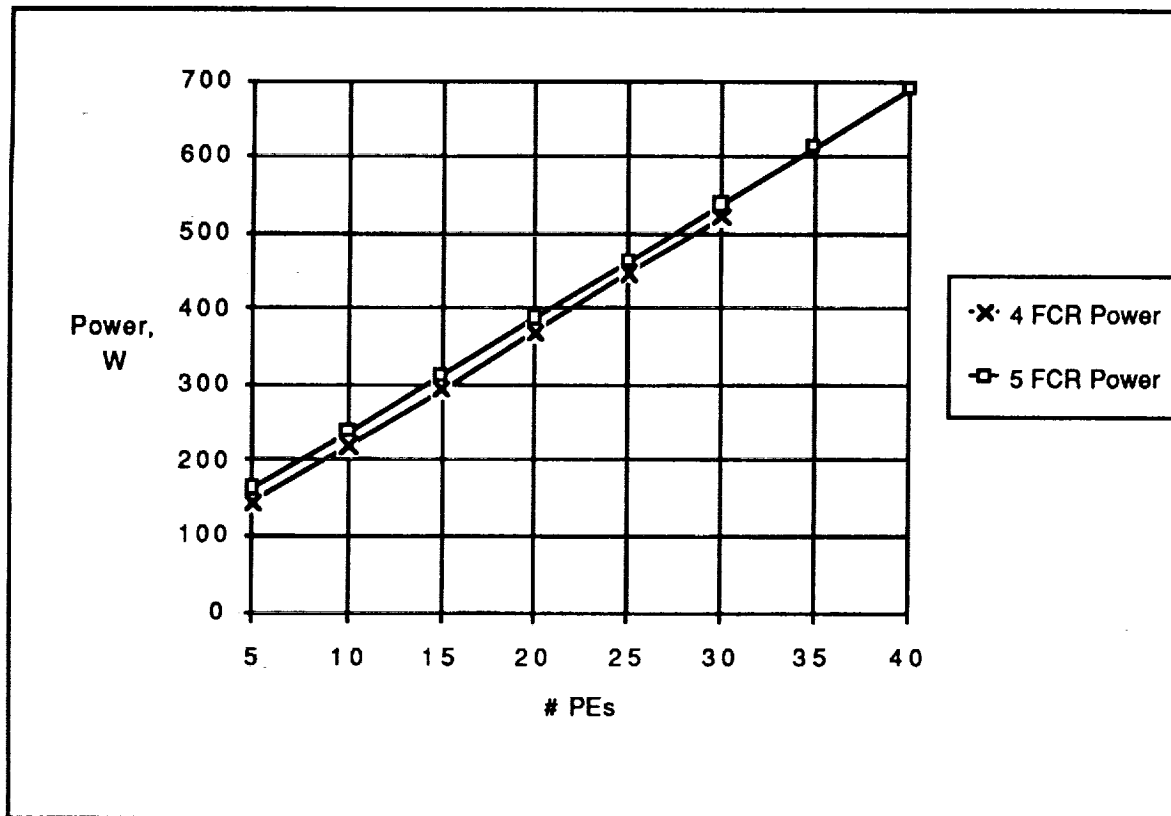


Figure 12-17. AFTA Power for Ground Vehicle Mission

12.2.3.5. Volume

The volumes of MIL-STD-344 AFTA modules are enumerated in Table 12-15.

PE Slots	1
NE Slots	1
PC slots	2
Rack Ends (# slots)	2
Slot Volume	3.55E-2 cu. ft. (10.5 x 7.3 x 0.80 in ³)

Table 12-15. AFTA Component Volumes for Ground Vehicle AFTA

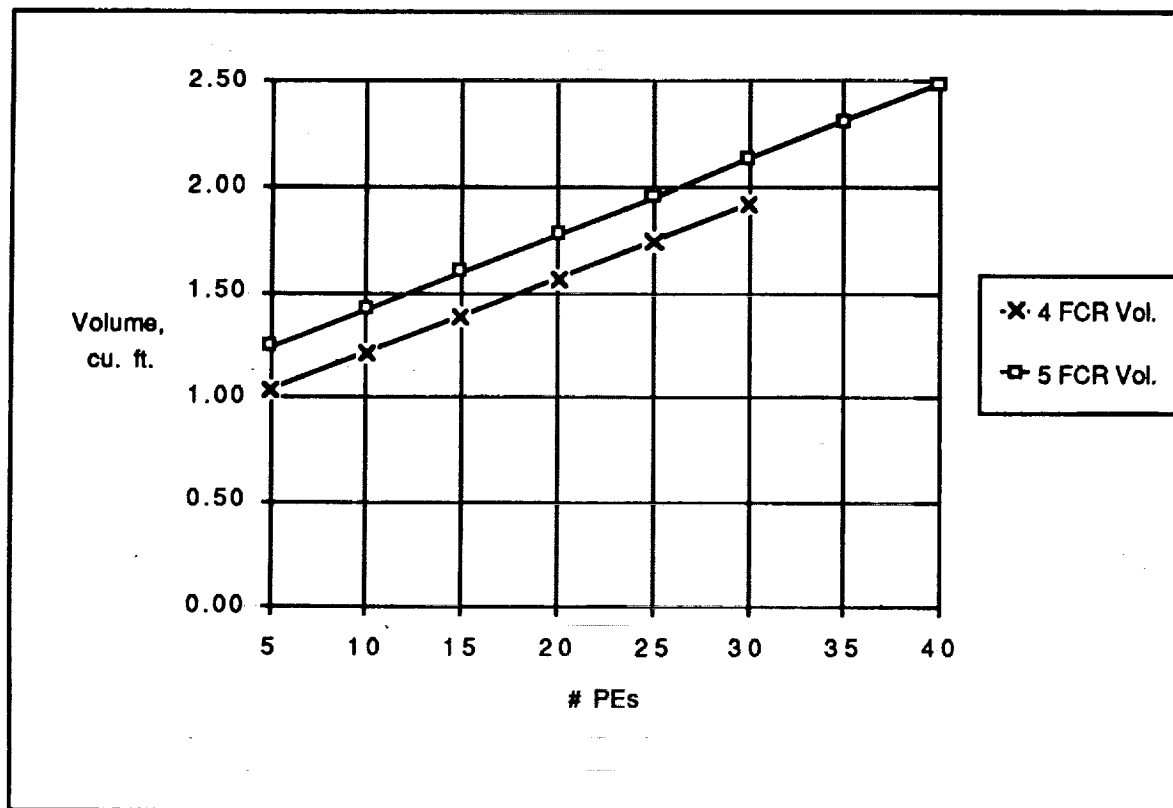


Figure 12-18. AFTA Volume for Ground Vehicle Mission

This page intentionally left blank.

Appendix A. References

- [Abl88] Abler, T., *A Network Element Based Fault Tolerant Processor*, MS Thesis, Massachusetts Institute of Technology, Cambridge, MA, May 1988.
- [AMD89a] *The SUPERNET Family for FDDI*, Advanced Micro Devices Data Book, Publication # 09734 Rev. C, February 1989.
- [AMD89b] *Am7968/Am7969-175 TAXIchipTM Integrated Circuits*, Advanced Micro Devices Data Sheet, Publication # 12834 Rev. A, November 1989.
- [ANSI139] "Fiber Distributed Data Interface (FDDI) - Token Ring Media Access Control (MAC)" American National Standard, ANSI X3.139-1987, November 5, 1986.
- [ANSI148] "Fiber Distributed Data Interface (FDDI) - Token Ring Physical Layer Protocol (PHY)," American National Standard, ANSI X3.148-1988, June 30, 1988.
- [ANSI166] "Fibre Data Distributed Interface (FDDI) - Token Ring Physical Layer Medium Dependent (PMD)," American National Standard, ANSI X3.166-1990, September 28, 1989.
- [APS90] Acarlar, M. S., Plourde, J. K., Snodgrass, M. L., "A High Speed Surface-Mount Optical Data Link for Military Applications," IEEE/AIAA/NASA 9th Digital Avionics Systems Conference Proceedings, October 15-18, 1990, p. 297-302.
- [Bab90] Babikyan, C., "The Fault Tolerant Parallel Processor Operating System Concepts and Performance Measurement Overview," *Proceedings of the 9th Digital Avionics Systems Conference*, October 1990, pp. 366-371.
- [Ber87] Bertsekas, D., Gallager, R., *Data Networks*, Prentice-Hall, 1987.
- [Ber90] Berger, K. M., Abramson, M. R., Deutsch, O. L., "Far-Field Mission Planning for Helicopters," CSDL Technical Report CSDL-R-2234, March 1990.
- [Bev90] Bevier, W.R., and Young, W.D., "The Proof of Correctness of a Fault-Tolerant Circuit Design," 2nd International Working Conference on Dependable Computing for Critical Applications, Tucson, AZ, February 1991.
- [Bic90] Bickford, M., and Srivas, M., "Verifying an Interactive Consistency Circuit: A Case Study in the Reuse of a Verification Technology," NASA Formal Methods Workshop 1990, NASA Conference Publication 10052, November 1990.

- [Biv88] Bivens, G. A., "Reliability Assessment of Surface Mount Technology (SMT)," RADC report RADC-TR-88-72, March 1988.
- [Bla91] Black, Uyless, OSI : A Model For Computer Communications Standards, Prentice-Hall, 1991.
- [Boo88] Booth, F., "Advanced Apache Architecture," 8th Digital Avionics Systems Conference, October 1988.
- [Bur89] Burkhardt, L., *Advanced Information Processing System: Local System Services*, NASA Contractor Report 181767, April 1989.
- [But88] Butler, R. W., "A Survey of Provably Correct Fault Tolerant Clock Synchronization Techniques," NASA Technical Report TM-100553, NASA Langley Research Center, February 1988.
- [CAMP] CAMP-1 Final Technical Report AFATL-TR-85-93, 3 Volumes, Available as DTIC AD-B102 654, AD-B102 655, and AD-B102 656 from Defense Technical Information Center, Alexandria, VA 22304-6145.
- [Car84] Carlow, G. D., "Architecture of the Space Shuttle Primary Avionics Software System", *Communications of the ACM*, 27(9):926-36, September 1984.
- [Cha84] Chambers, F. B., ed., *Distributed Computing*, Academic Press, 1984.
- [Che87] Cheng, S-C., Stankovic, J. A., Ramamritham, K., "Scheduling Algorithms for Hard Real-Time Systems - A Brief Survey," in *Hard Real-Time Systems*, *IEEE Computer Society Press*, 1988.
- [Coh87] Cohn, Marc D., "The Conformance of the ANSI FDDI Standard to the SAE-9B HART High Speed Data Bus Requirements for Real-Time Local Area Networks," Society of Automotive Engineers Aerospace Systems Conference Proceedings, November 1987.
- [Coh88] Cohn, Marc D., "The Fiber Optic Data Distribution Network: A Network for Next-Generation Avionics Systems," AIAA/IEEE 8th Digital Avionics Systems Conference Proceedings, October 17-20, 1988, p. 731-737.
- [Coh90a] Cohen, G. C., et. al., *Design of an Integrated Airframe/Propulsion Control System Architecture*" NASA Contractor Report 182004, March 1990.
- [Coh90b] Cohen, G. C., et. al., *Final Report: Design of an Integrated Airframe/Propulsion Control System Architecture*, NASA Contractor Report 182007, March 1990.
- [Cohn88] Cohn, A., "Correctness Properties of the Viper Block Model: The Second Level," Tech. Report 134, Univ. of Cambridge, Cambridge, England, May 1988.

- [Com91] Comer, D. E., Internetworking with TCP/IP, Prentice-Hall, 1991.
- [CSDL9214] Completion of the Advanced Information Processing System, response to NASA Langley Research Center, CBD Announcement REF SS017, issue PSA-9214, November 12, 1986.
- [Cullyer 88] Cullyer, W. J., "Implementing Safety-Critical Systems: The VIPER Microprocessor," VLSI Specification, Verification and Synthesis, Kluwer Academic Publishers, 1988.
- [CVC2] "System Specification for Combat Vehicle Command and Control (DRAFT)," CVC2 Systems Implementation Working Group, 31 October 1990.
- [DACS] Defense & Analysis Center for Software, Kaman Sciences Corporation, P.O. Box 120, Utica, NY 13503.
- [Dal73] Daly, W.M., Hopkins, A.L., and McKenna, J.F., "A Fault-Tolerant Digital Clocking System," 3rd International Symposium on Fault Tolerant Computing, Palo Alto, CA, June 1973.
- [Deu88] Deutsch, O. L., Desai, M., "Development and Demonstration of an On-Board Mission Planner for Helicopters," CSDL Technical Report CSDL-R-2056, April 1988.
- [DID80811] "VHSIC Hardware Description Language (VHDL) Documentation," Data Item Description, DD Form 1664, DI-EGDS-80811, May 11, 1989.
- [DiV90] Di Vito, B. L., Butler, R. W., Caldwell, J. L., *Formal Design and Verification of a Reliable Computing Platform for Real-Time Control*, NASA Technical Memorandum 102716, October 1990.
- [DiV91] Di Vito, B., Butler, R., and Caldwell, J., "High Level Design Proof of a Reliable Computing Platform," 2nd International Working Conference on Dependable Computing for Critical Applications, Tucson, AZ, February 1991.
- [Dol82] Dolev, D., "The Byzantine Generals Strike Again," *Journal of Algorithms*, Vol. 3, 1982, pp. 14-30.
- [Dol84] Dolev, D., Dwork, C., Stockmeyer, L., "On the Minimal Synchronism Needed for Distributed Consensus," IBM Research Report RJ 4292 (46990), 5/8/84.
- [Fel90] Felter, S. C., Douglas, P. H., Smith, C. A., "Avionics System Integration for the MH-53J Helicopter," 9th Digital Avionics Systems Conference, October 1990.
- [Fis82] Fischer, M. J., Lynch, N. A., "A Lower Bound for the Time to Assure Interactive Consistency," *Information Processing Letters*, Vol. 14, No. 4, 13 June 1982, pp. 183-186.

- [Foh89] Fohler, G., Koza, C., "Heuristic Scheduling for Distributed Real-Time Systems," Research Report No. 6/89, Institut für Technische Informatik, Technische Universität Wien, Vienna, Austria, April 1989.
- [Gal90] Galetti, R. R., Real-Time Digital Signatures and Authentication Protocols, Master of Science thesis, Massachusetts Institute of Technology, May 1990.
- [Goe91] Goel, A.L., and Sahoo, S.N., "Formal Specifications and Reliability: An Experimental Study," 1991 International Symposium on Software Reliability Engineering, Austin, Texas, May 1991.
- [Gua90] Guaspari, D., Marceau, C., and Polak, W., "Formal Verification of Ada Programs," IEEE Transactions on Software Engineering, Special Issue on Formal Methods in Software Engineering, Vol. 16, No. 9, September 1990.
- [Han89] Hanaway, J. F., Morrehead, R. W., *Space Shuttle Avionics System*, NASA SP-504, 1989.
- [Har87] Harper, R., *Critical Issues in Ultra-Reliable Parallel Processing*, PhD Thesis, Massachusetts Institute of Technology, Cambridge, MA, June 1987.
- [Har88a] Harper, R., Lala, J., Deyst, J., "Fault Tolerant Parallel Processor Overview," *18th International Symposium on Fault Tolerant Computing*, June 1988, pp. 252-257.
- [Har88b] Harper, R., "Reliability Analysis of Parallel Processing Systems," *Proceedings of the 8th Digital Avionics Systems Conference*, October 1988, pp. 213-219.
- [Har91a] Harper, R., Lala, J., *Fault Tolerant Parallel Processor*, J. Guidance, Control, and Dynamics, V. 14, N. 3, May-June 1991, pp. 554-563.
- [Har91b] Harper, R., Alger, L., Lala, J., "Advanced Information Processing System: Design and Validation Knowledgebase," NASA Contractor Report 187544, September 1991.
- [Hir90] Hird, G.R., "Formal Methods in Software Engineering," 9th AIAA/IEEE Digital Avionics Systems Conference, Virginia Beach, VA, October 1990, pp. 230-234.
- [Hun86] Hunt, W.A., "FM8501: A Verified Microprocessor," Proceedings of IFIP Working Group 10.2 Workshop, North Holland, Amsterdam, 1986.
- [Hwa84] Hwang, K., Briggs, F., Computer Architecture and Parallel Processing, McGraw-Hill, 1984.
- [IEEE1076] "VHDL Language Reference Manual," IEEE Standard, IEEE Std 1076-1987, March 31, 1988.

- [IEEE8021] "Local and Metropolitan Area Networks: Overview and Architecture," IEEE Standard, IEEE Std 802-1990, May 31, 1990.
- [IEEE8022] "Logical Link Control," IEEE Standard, IEEE Std 802.2-1989, August 17, 1989.
- [IEEE8023] "Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications," IEEE Standard, IEEE 802.3-1988, June 9, 1988.
- [IEEE8024] "Token-Passing Bus Access Method and Physical Layer Specifications," IEEE Standard, IEEE 802.4-1990.
- [J88N2] "Linear Token Passing Multiplex Data Bus Protocol," Joint Integrated Avionics Working Group Standard, Document J88-N2,
- [J8701] "Advanced Avionics Architecture (A3) Standard," Joint Integrated Avionics Working Group Standard, Document J87-01.
- [Klj89] Kljaich, J., Jr., Smith, B.T., and Wojcik, A.S., "Formal Verification of Fault Tolerance Using Theorem-Proving Techniques," *IEEE Transactions on Computers*, Vol. 38, No. 3, March 1989.
- [Kop89] Kopetz, H., et. al., "Distributed Fault-Tolerant Real-Time Systems: The MARS Approach," *IEEE Micro*, 9(1):25-40, February 1991.
- [Kop91] Kopetz, H., et. al., "The Rolling Ball on MARS," Institut fur Technische Informatik Research Report No. 13/91, Technische Universitat Wien, Vienna, Austria, November 1991.
- [Kri85] Krishna, C. M., Shin, K. G., Butler, R. W., "Ensuring Fault Tolerance of Phase Locked Clocks," *IEEE Trans. Computers*, Vol. C-34, No. 8, August, 1985.
- [Lal84] Lala, J. H., "An Advanced Information Processing System," 6th AIAA-IEEE Digital Avionics Systems Conference, Baltimore, MD, Dec. 1984.
- [Lal84] Lala, J. H., "An Advanced Information Processing System," 6th AIAA-IEEE Digital Avionics Systems Conference, Baltimore, MD, December 1984.
- [Lal85] Lala, J. H., "Advanced Information Processing System: Fault Detection and Error Handling," AIAA Guidance, Navigation and Control Conf., Snowmass, CO, Aug. 1985.
- [Lal86a] Lala, J.H., "Fault Detection, Isolation, and Reconfiguration in the Fault Tolerant Multiprocessor," *Journal of Guidance, Control, and Dynamics*, Sept-Oct. 1986.
- [Lal86b] Lala, J. H., "A Byzantine Resilient Fault Tolerant Computer for Nuclear Power Plant Applications," 16th Annual International Sym-

posium on Fault Tolerant Computing Systems, Vienna, Austria, 1-4 July 1986.

- [Lal89] Lala, J.H., et. al., "Study of a Unified Hardware and Software Fault Tolerant Architecture," NASA Contractor Report 181759, January 1989.
- [Lal91] Lala, J.H., R. Harper, K. Jaskowiak, G. Rosch, L. Alger, and A. Schor "AIPS for Advanced Launch System: Architecture Synthesis Report", NASA Contractor Report 187544, September 1991.
- [Lam85] Lamport, L., Melliar-Smith, P. M., "Synchronizing Clocks in the Presence of Faults," *Journal of the ACM*, 32(1):52-78, January 1985.
- [Lap90] "Dependability: Basic Concepts and Terminology," J.C. Laprie - Editor, Published by International Federation for Information Processing (IFIP) Working Group 10.4 on Dependable Computing and Fault Tolerance, December 1990.
- [Leh87] Lehoczky, Sha, Ding, *The Rate Monotonic Scheduling Algorithm - Exact Characterization and Average Case Behavior*, Technical Report, Department of Statistics, Carnegie-Mellon University, 1987.
- [Liu73] Liu, C. L., Layland, J. W., "Scheduling Algorithms for Multiprogramming in a hard Real-time Environment," *J. ACM*, 20(1):46-61, 1973.
- [LSP82] Lamport, L., Shostak, R., Pease, M., "The Byzantine Generals Problem," *ACM Transactions on Programming Languages and Systems*, Vol. 4, No. 3, July 1982, p. 382-401.
- [MA-HDBK] Modular Avionics Handbook, Document No. 21530(0-6), FSCM 51993, Draft C, U. S. Air Force ASD-ALD/AX, 19 April 1990.
- [Ma78] Martin, D. L., Gangsaas, D., "Testing of the YC-14 Flight Control System Software," *AIAA Journal of Guidance, Control, and Dynamics*, Vol. 1, No. 4, July-August 1978.
- [McE88] McElvany, M. C., "Guaranteeing Deadlines in MAFT," *IEEE Real-Time Systems Symposium*, Huntsville, AL, December 1988.
- [MIL-HDBK-0036] "Survivable Adaptable Fiber Optic Embedded Network II - SAFENET II," Military Handbook, MIL-HDBK-0036, 1 March, 1990.
- [MIL-HDBK-59] MIL-HDBK-59, "Computer-Aided Acquisition and Logistic Support (CALS) Program Implementation Guide," 20 December 1988.
- [MIL-HDBK-217E] MIL-HDBK-217E, "Reliability Prediction of Electronic Equipment," 2 January 1990.
- [MIL-STD-344] MIL-STD-344 (draft), "Standard Army Vetrionics Architecture," 14 September, 1990.

- [MIL-STD-785B] MIL-STD-785B, "Reliability Program for Systems and Equipment Development and Production," 15 September 1980.
- [MIL-STD-1553] "Aircraft Internal Time Division Command/Response Multiplex Data Bus," Military Standard, MIL-STD-1553B, 12 February, 1980.
- [MIL-STD-1815A] MIL-STD-1815A, "Reference Manual for the Ada Programming Language," 17 February 1983.
- [NAS1-18565-14] Statement of Work for NASA Contract NAS1-18565, Task 14, June 1990.
- [Osd88] Osder, S. S., "Digital Fly-by-Wire System for Advanced AH-64 Helicopters," 8th Digital Avionics Systems Conference, October 1988.
- [Pe80] Pease, M., Shostak, R., Lamport, L., "Reaching Agreement in the Presence of Faults," *Journal of the ACM*, Vol. 27, No. 2, April 1980, pp. 228-234.
- [PEI90120] *XTP® Protocol Definition, Revision 3.5*, Published by Protocol Engines Inc., September 1990.
- [Pek88] Pekelsma, N. J., "Optimal Guidance with Obstacle Avoidance for Nap-of-the Earth Flight," NASA Contractor Report 177515, December 1988.
- [Pus89] Puschner, P., Koza, C., "Calculating the Maximum Execution Time of Real-Time Programs," *Real-Time Systems*, 1(2):159-176, September 1989.
- [Rad90] PMV 68 CPU-3A Specification, Issue 3, Publication No. 681/SA/04085, Radstone Technology plc, 1990.
- [Rus89] Rushby, J., von Henke, F., "Formal Verification of a Fault Tolerant Clock Synchronization Algorithm," NASA Contractor Report 4239, June 1989.
- [SAE91] SAE/AS-2A Subcommittee RTMT Statement on Requirements for Real-Time Communication Protocols (RTCP), Issue #1, SAE ARD50007, August 2 1991.
- [San90] STAR MVP Technical Description, Document No. 4069718, Lockheed Sanders, 25 June 1990.
- [Sch91] Schutz, W., "On the Testability of Distributed Real-Time Systems," *Proc. Tenth Symposium on Reliable Distributed Systems*, Pisa, Italy, September, 1991.
- [Spi89] Spivey, J.M., *The Z Notation. A Reference Manual*, Prentice Hall International (UK) Ltd, 1989.
- [Spi90] Spivey, J.M., "Specifying a Real-Time Kernel," *IEEE Software*, Special Issue on Formal Methods, Vol. 7, No. 5, Sep 1990.

- [Sri90] Srivas, M. and Bickford, M., "Formal Verification of a Pipelined Microprocessor," IEEE Software, Special Issue on Formal Methods, Vol. 7, No. 5, September 1990.
- [Sta87] Stankovic, J. A., Ramamritham, K., "The Design of the Spring Kernel," *Proc. of the Real Time Systems Symposium*, December 1987.
- [Sun74] Sundstrom, R. J., "On-Line Diagnosis of Sequential Systems," PhD Thesis, University of Michigan, 1974.
- [Tan88] Tanenbaum, A. S., Computer Networks, second edition, Prentice-Hall, 1988.
- [X3T95] "FDDI Station Management (SMT)," Preliminary Draft Proposed American National Standard, X3T9.5/84-49, Rev. 6.2, May 18, 1990.

Xpress Transfer Protocol®, XTP®, and Protocol Engine® are registered trademarks of Protocol Engines, Incorporated.

Appendix B. Glossary of Terms and Acronyms

AFTA-Army Fault-Tolerant Architecture-A computer designed for both high reliability and high throughput. The AFTA is based on the FTPP architecture.

aperiodic tasks-A set of tasks whose iteration rates are unknown or undefined.

ASIC-Application Specific Integrated Circuit-A type of integrated circuit that can be custom designed by the hardware engineer so that it will perform a particular logic or processing function and at the same time save circuit board space and power consumption. The advent of VLSI design techniques has made ASICs a more flexible and practical option for hardware designers.

ATP-Authentication Protocol-A protocol utilized by the BRNP to sign outgoing packets and to test the authenticity of incoming packets.

ATPG-Automatic Test Pattern Generation-The generation of test vectors directly from a netlist for verification of device functionality. Test vectors from an ATPG program do not test the correct functionality of the device; they only test that the device is a correct implementation of the design as specified by the netlist.

behavioral VHDL is defined to be a VHDL architecture which uses any of the legal VHDL constructs, including those which do not correspond to possible hardware realizations of the description (i.e., pure behavioral may not be synthesizable). A level of description that specifies a device functionally in terms of output reactions to input stimulus. A behavioral description can also specify the timing relationships of inputs to outputs.

BIT-Built In Test-This is an internal diagnostic testing system that is included as part of the AFTA design. There are three forms of the BIT-- I-BIT is the initial power-on test system, M-BIT is for maintenance testing, C-BIT is the continuous in-flight test system.

BRNP-Byzantine Resilient Network Protocol-A network layer protocol which implements the Byzantine Resilient Virtual Circuit in order to guarantee that all messages are delivered accurately.

broadcast addressing-A method of station addressing using an identifier that causes all stations to respond to the specified address.

bypass-The ability to effectively isolate a node from the network without disrupting the continuity of the network.

Byzantine Resilient-Capable of tolerating Byzantine faults. A Byzantine Resilient system is capable of handling arbitrarily malfunctioning components that may supply faulty information to other parts of the system thereby causing a spread of faulty information within the system.

C3-Cluster 3-An FTTP model number. Composed of either 4 or 5 FCRs, 3-40 processors, 1-40 VIDs, simplex, triplex, and quadruplex processor redundancy levels. Previous FTTP models were C1 (4 FCRs, 16 processors, 4-16 VIDs, simplex, duplex, triplex, and quadruplex processor redundancy levels) and C2 (4 FCRs, 4 processors, one fixed quad VID).

cache-A form of memory that is typically much faster and much smaller than main memory. Through utilization of cache memory, a processor's throughput will be increased. Typically cache memory acts as a staging area for data; information will be pulled from main memory and temporarily stored in cache while it undergoes processing.

CDU-Cockpit Display Unit-A cathode ray tube display located in the vehicle cockpit for display of system status. The CDU may display overall AFTA system status, LRU level status, or LRM level status.

CID-Communication Identification-A designation assigned to each task which is used for intertask communication.

class test-A test of the Network Element voting mechanism that requests a non-congruent message exchange selectively on each channel of a fault masking group.

cluster-An FTTP consisting of 4 or 5 FCRs containing at least one virtual processing site. Multiple clusters could be connected by a network device (such as a fault-tolerant data bus) to provide even greater throughput than a single cluster. Most references to an FTTP refer to a single cluster design.

CME-Common Mode Fault-A type of malfunction which will cause multiple faults or complete execution failure within a redundant processing group. Common mode faults may result from software flaws, hardware bugs, design flaws, massive electrical upsets etc.

concurrent I/O-Input/Output processes that allow the associated virtual group to perform other tasks while I/O is collecting data. This allows for greater processor throughput.

CRC-Cyclic Redundancy Check-An error detecting code used in data communications that allows the unit receiving a message to ensure through binary mathematics that it is the same message sent by the transmitting unit.

CSMA/CD-Carrier Sense Multiple Access with Collision Detection-A form of media access control whereby a potential transmitting station will monitor the bus to ensure that it is clear before transmission begins. During transmission, the station also monitors the bus to check for message collisions. If a collision occurs, the message must be re-transmitted.

CT-Configuration Table-A table stored on the Network Element that contains the current configuration of the system, i.e. which processors are members of which virtual groups.

DAIS-Digital Avionics Instruction Set-A benchmark for measuring processor throughput.

depot test-A set of diagnostic level tests executed outside of the constraints of a real-time environment with emphasis on the isolation of chip level faults in these components. These tests would occur at a maintenance repair facility in contrast to the various forms of built-in testing.

DPRAM-Dual-Port Random Access Memory-The type of memory that occupies the data segment. It provides a buffer between the NE and the PE; both the NE and the PE may access the data segment asynchronously, provided that they do not attempt to access the same location.

DR-Discrepancy Report-A report that is filed whenever unexpected behavior of the hardware, software, or system is encountered. By recording observable symptoms of the system throughout testing, integration, verification and validation, one may better trace and identify system flaws.

entity-A specific instance of a protocol element in an Open Systems Interconnection layer or sublayer.

FCR-Fault Containment Region-Usually comprised of a number of line replaceable modules such as Processing Elements, Network Elements, input/output controller, and power conditioners. The AFTA is made up of four or five FCR's, and each FCR usually resides

on a single circuit board (with the exception of the power conditioner). An interchangeable term for the FCR is Line Replaceable Unit or LRU.

FDDI-Fiber Distributed Data Interface-A networking standard developed by the American National Standards Institute to provide high bandwidth for Local Area Networks.

FDIR-Fault Detection, Identification and Recovery-FDIR software designed for the AFTA allows it to sustain multiple successive faults by identifying a faulty component and reconfiguring the AFTA system operation to compensate for the fault.

FIFO-First In First Out-A type of information buffer in which the data that is stored first chronologically will be the first to be extracted.

FMEA-Failure Modes and Effects Analysis

FMG-Fault Masking Group-A logical grouping of three or four processors to enhance the reliability of critical tasks. The members of an FMG execute the same code with the same data and periodically exchange messages to ensure that they produce the same outputs.

ETC-Fault Tolerant Clock-A distributed digital phase-locked loop used for synchronization of AFTA fault containment regions.

ETDB-Fault Tolerant Data Bus-A local area network designed around principles of Byzantine resilience. Its primary objective is to provide an optimal internetworking system between simplex and redundant processing sites.

ETNP-Fault Tolerant Navigation Processor-The initial ground vehicle application for the AFTA is for the navigations system in Armored Systems Modernization vehicles.

FTPP-Fault-Tolerant Parallel Processor-A computer designed for both high reliability and high throughput. The core of the FTTP is the Network Element.

functional reliability-The probability that a given function can be executed because its resources are operational.

functional synchronization-In maintaining synchronous operation, the members of a VID perform a synchronizing act after some sequence of functions has been completed. The sequence of functions between the synchronization points is referred to as a frame.

GC-Global Controller-A microcoded finite-state machine used to coordinate the functions throughout the Network Element.

graceful degradation-Through self-testing, a virtual group may identify a faulty member and gracefully degrade its redundancy level using a configuration table update message to eliminate the faulty channel.

IOC-Input/Output Controller-These devices connect the AFTA to the outside world, and they must be compatible with the bus connecting elements of the FCR. They may have a programmable processor on board to drive the I/O, or they may require off-board processors for operation.

IPS-Instructions Per Second-The number of machine language instructions that a processor will execute every second. This measurement is used to reference the speed of the processor.

ISO/OSI-International Standards Organization/Open Systems Interconnection-A specification and model for computer communication networks.

LAN-Local Area Network-A network topology that interconnects computer systems separated by relatively short distances (2-2000 meters). LAN technology is usually based on a shared medium with no intermediate switching nodes required.

leaf-level-(VHDL) The models at the bottom of the model tree. Leaf-level models in VHDL are always pure behavioral models.

LERP-Local Exchange Request Pattern-A string of bytes describing the current state of the input and output buffers for each processor in an FCR. The LERP is used to generate the SERP. Each FCR has a different configuration, therefore the LERPs for each FCR will be different. For this reason, LERPs must be treated as single-source data.

link-An element in a physical network that provides interconnection between nodes.

LOC-Loss of Control-This will occur as a result of a failure in any flight critical portion of the Flight Control System. For analysis purposes, LOC will be considered as a total loss of the vehicle.

Local FDI-Each virtual group will exercise its own fault detection and identification processes to monitor failures among its processors. Also, each virtual group may initiate its own recovery options.

logical addressing-A method of station addressing using an identifier that may select a group of stations to respond to the specified address.

LRM-Line Replaceable Module-The physical unit for field diagnosis and repair. Typically it consists of one circuit card assembly with one or more Processing Elements.

LTPB-Linear Token Passing Bus-A media access control method whereby stations pass a token along a virtual ring from one to another. A station may only transmit when it possesses the token.

MDC-Minimum Dispatch Complement-This specifies the absolute minimum level of operability for the AFTA system to be cleared for a sortie.

media access control-The method by which access to the physical network media is limited to a single node so that communications over the media are undisturbed.

media layer-One or more physical layer media. Multiple media layers are physically and electrically isolated from each other to the same degree as a fault-containment region in a fault-tolerant computer. Most traditional LANs use only a single network layer. A Byzantine resilient network usually employs multiple media layers for redundancy.

memory alignment-A process whereby the RAM and registers in each processor of a virtual group are made congruent as part of the resynchronization of a virtual group.

mission reliability-Arithmetically speaking, mission reliability is one minus the probability that failure of the AFTA causes abortion of the mission.

MMC-Minimum Mission Complement-This specifies the minimum level of AFTA operability for the vehicle to continue its mission.

NDI-Non-Developmental Item

NE-Network Element-The hardware device which provides the connectivity between virtual groups. The primary function of the NE is to exchange and vote packets of data pro-

vided by the processors. The ensemble of Network Elements forms a virtual bus network to which all virtual groups are connected.

NEID-Network Element ID-The name by which a Network Element is known in the physical AFTA configuration. An NEID refers to a specific Network Element in the system, i.e. the same NEID on different FCRs refers to the same Network Element. The NEID is also used to refer to the FCR in which the referenced Network Element resides. By convention, letters are used to denote the NEID.

netlist-A list defining interconnections of components. Netlists are typically used for designing printed circuit boards or ASICs.

NIU-Network Interface Unit-The connection between a station and the FTDB

node-An element in a physical network that provides the necessary interface between a station and the network media.

nonpreemptible I/O dispatcher-A task on the virtual group that manages the execution of certain I/O instructions that cannot be interrupted.

packet-A block of data consisting of a header, data, and a trailer exchanged between peer protocol entities. The term packet is somewhat generic and is applied at all levels of the protocol hierarchy.

packet-A string of data of fixed or variable length for transmission from one processor to another through an inter-processor network. A message-passing network handles data in packets. The term packet is used here to refer to a fixed-size (64 bytes) block of data which is transmitted by the Network Elements.

PDU-Protocol Data Unit-A fancy name for a packet. PDU is the name used by OSI.

PE-Processing Element-A hardware device which provides a general or special purpose processing site. A minimal PE configuration contains a single processor and local memory (RAM and ROM). PEs may optionally have private I/O, making them a combination PE and IOC.

PEID-Processing Element ID-The name by which a Processing Element is known in the physical AFTA configuration. Each PE in an FCR has a unique PEID. However, the same

PEID may be used by another processor in another FCR. A combination of NEID and PEID is used to uniquely identify a single Processing Element within a cluster.

physical addressing-A method of station addressing using a unique identifier such that at most one station responds to the specified address.

PIMA-Portable Intelligent Maintenance Aid-A system resembling a laptop computer which will initiate the maintenance built in testing (M-BIT), interrogate AFTA for fault information logged during a mission, and extract maintenance records for system components.

PMD-Physical layer Medium Dependent-The standard which defines the physical medium that is used for the data communications channel on a network.

presence test-The polling of various components to determine if each is active and synchronized. The testing may be performed on members of virtual groups or on the virtual groups themselves.

primitive-A function or procedure that one entity provides to another. The primitive definition specifies the inputs, outputs, and data formats for the primitive.

PROM-Programmable Read Only Memory-A form of computer memory that will store a permanent copy of one or more subroutines specifically intended for use by a particular microprocessor. PROM's allow for a certain level of hard-wired software control over the processor.

quadruplex-A virtual group consisting of four processing sites.

rate group dispatcher-An RG4 task that is responsible for controlling the execution of the rate group tasks and providing reliable communication between the rate group tasks throughout the system.

Register Transfer Level (RTL) VHDL-A behavioral format which specifies the functionality of a block from the standpoint of random combinational logic and/or synchronous registers. For the purpose of the AFTA NE development, RTL is defined to be synthesizable behavioral VHDL, that is, a behavioral VHDL description that is suitable for input to a synthesis tool.

reprocurement-The act of obtaining new parts to replace parts in an existing system, or to build additional copies of an existing design.

RG-Rate Group-A set of tasks whose iteration rate is well-defined and whose execution times do not exceed the iteration frame (the inverse of the iteration rate).

RISC-Reduced Instruction Set Computer-A type of microprocessor which utilizes a limited set of machine language instructions to allow for more rapid execution of those instructions and thus greater throughput for the computer.

RTS-Run Time System

SAVA-Standard Army Vetrronics Architecture

sequential I/O-Input/Output processes that require the managing virtual group to completely supervise the activity. In other words, the virtual group must block itself until the I/O is finished.

SERP-System Exchange Request Pattern-A string of bytes describing the current state of the input and output buffers for each processor in the system. The SERP is used to determine if packets can be sent from one virtual group to another. The LERP from each FCR is exchanged using a source congruency to generate the SERP. Because the SERP originates from a source congruency exchange, it can be considered congruent throughout all functioning FCRs.

SIFT-Software Implemented Fault Tolerance-System fault tolerance functions achieved primarily through operating system programming rather than primarily through dedicated hardware.

simplex-A virtual group consisting of only one processing site.

single-source data-An element of information which originates from a single point. Examples of single-source data include sensor readings, input values, and syndromes. Single-source data must be distributed to fault-masking groups using a source congruency exchange to maintain Byzantine resilience.

sortie availability-One minus the probability that the vehicle is prevented by the AFTA from beginning a mission at the desired time.

source congruency-A type of exchange used to distribute data from a single source, such as an input device, to members of a fault-masking group. The source congruency, which is

also known as a class 2, 2-round exchange, or interactive consistency, is a primary requirement for a Byzantine resilient system.

station-A device connected to a network that can transmit or receive data over the network. Often a station is a processing site. In the FTDB, a station can be a redundant processing site.

structural VHDL-A level of description that specifies a VHDL architecture by defining interconnections of instantiations of VHDL entities . A structural description resembles a conventional netlist.

syndrome-A bit field indicating the observance of unusual behavior somewhere in the system. Syndromes can be used in an attempt to diagnose and repair faults in the system.

System FDI- A process that will coordinate system status and fault information as well as testing and analyzing shared components.

task migration-The movement of a necessary task from a failed processor to another processor within the same fault containment region.

test bench-A model of a test fixture that is used to test a device being designed with VHDL. The test bench is written in VHDL and provides a non-proprietary way of stimulating and monitoring a design in a simulator.

testability-The ability to unambiguously ascertain the functionality of each Line Replaceable Module of the AFTA.

TF/TA/NOE-Terrain Following/Terrain Avoidance/Nap of the Earth-A typical helicopter mission application for which the AFTA will be designed.

THT-Token Holding Timer-A method used with token passing media access protocols to limit the amount of time each station can transmit on the network.

timeout-A value of time used to monitor skew between processors of an FMG. All processors in an FMG should be synchronized to within one timeout value, so if a processor does not respond within the timeout period, that processor is considered faulty, and the other processors will continue uninhibited. Timeouts are necessary on the AFTA to prevent faulty processors from halting the system.

timestamp-A 32-bit quantity that indicates the relative time within the cluster. The Network Element places a timestamp in the input info block for each packet successfully delivered to a virtual group.

TNR-Transient NE Recovery-The procedure by which a Network Element which has suffered a transient fault is reintegrated into the cluster. The first part of TNR is similar to the ISYNC procedure. TNR also specifies the realignment of the Network Element state.

transient recovery policy-A recovery option whereby the faulty component is immediately disabled and an attempt is made to reintegrate the component into the system.

triplex-A virtual group consisting of three processing sites.

validation-The process of demonstrating that an implemented system correctly performs its intended functions under all reasonably anticipated operational scenarios.

validity-In a Byzantine resilient system, a condition in which all functioning members of a fault-masking group are guaranteed to possess correct data. The validity condition also implies the agreement condition.

vehicle reliability-One minus the probability that the vehicle is lost due to failure of the AFTA.

VG-virtual group-A grouping of one or more processors to form a virtual (possibly redundant) single processing site. All processors in a virtual group execute the same instruction stream. If a virtual group has more than one member, those members must reside in different FCRs. Virtual groups of 3 or more members are known as fault-masking groups.

VHDL-VHSIC Hardware Description Language-A language for specifying hardware design. VHDL designs can be expressed in a behavioral or a structural method. VHDL also defines a simulation environment and incorporates an intrinsic sense of time.

VHSIC-Very High Speed Integrated Circuit-A Government-funded project to develop technologies to be applied to new, high speed integrated circuits. The VHSIC Hardware Description Language (VHDL) was developed under the VHSIC program.

VID-Virtual Identifier-The name by which a virtual group is known to the system. Also, sometimes used as a synonym for virtual group.

voted message-A message sent by all members of a redundant processing group. This message type is only used when exact consensus among all redundant members is expected. This is also known as a Class 1 message.

voter test-A test of the Network Element voting mechanism that seeds non-congruent values selectively on each channel of a fault masking group.

WAN-Wide Area Network-A network topology that interconnects computer systems separated by long distances. WAN systems usually use packet switched technology.

watchdog timer-A simple timekeeper that will monitor operations in both the Processing Elements and the Network Elements to keep the hardware and software from wandering into undesirable states.

working group-The set of FCRs in a cluster which are synchronized and in the operational phase. An FCR which suffers a fault drops out of the working group. The working group may attempt to reintegrate the failed FCR into the working group.

WPV-Weight Power Volume-These are physical characteristics used to describe the AFTA.

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.				
1. AGENCY USE ONLY (Leave blank)		2. REPORT DATE July 1992		3. REPORT TYPE AND DATES COVERED Contractor Report
4. TITLE AND SUBTITLE Advanced Information Processing System: The Army Fault Tolerant Architecture Conceptual Study - Volume II: Army Fault Tolerant Architecture Design and Analysis			5. FUNDING NUMBERS WU 505-64-52-53 C NAS1-18565 TA 14	
6. AUTHOR(S) R. E. Harper, L. S. Alger, C. A. Babikyan, B. P. Butler, S. A. Friend, R. J. Ganska, J. H. Lala, T. K. Masotto, A. J. Meyer, D. P. Morton, G. A. Nagle, and C. E. Sakamaki				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) The Charles Stark Draper Laboratory, Inc. 555 Technology Square Cambridge, MA 02139			8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) National Aeronautics and Space Administration Langley Research Center Hampton, VA 23665-5225			10. SPONSORING / MONITORING AGENCY REPORT NUMBER NASA CR-189632, Volume II	
11. SUPPLEMENTARY NOTES Technical Monitor: Carl R. Elka, Aerostructures Directorate, AVRADA, AVSCOM, Langley Research Center, Hampton, VA				
12a. DISTRIBUTION / AVAILABILITY STATEMENT Unclassified- Unlimited Star Category 62			12b. DISTRIBUTION CODE	
13. ABSTRACT (Maximum 200 words) The Army Avionics Research and Development Activity (AVRADA) is pursuing programs that would enable effective and efficient management of large amounts of situational data that occurs during tactical rotorcraft missions. The "Computer-Aided Low Altitude Night Helicopter Flight Program" has identified automated Terrain Following/Terrain Avoidance, Nap of the Earth (TF/TA, NOE) operation as key enabling technology for advanced tactical rotorcraft to enhance mission survivability and mission effectiveness. The processing of critical information at low altitudes with short reaction times is life-critical and mission critical necessitating a ultrareliable/high throughput computing platform for dependable service for flight control, fusion of sensor data, route planning, near-field/far field navigation, and obstacle avoidance operations. To address these needs the Army Fault-Tolerant Architecture (AFTA) is being designed and developed. This computer system is based upon the Fault-Tolerant Parallel Processor (FTPP) developed by Charles Stark Draper Laboratory, Inc. (CSDL). AFTA is hard real-time, Byzantine fault-tolerant parallel processor which is programmed in the ADA language. This document describes the results of conceptual study (Phase I of a 3-year project) of the AFTA development. This document contains detailed descriptions of the program objectives, the TF/TA NOE application requirements, architecture overview, hardware design, operating systems design, analytical models and development plan.				
14. SUBJECT TERMS Fault-tolerant, Real-time digital computer, Terrain-following/terrain avoidance helicopter operation			15. NUMBER OF PAGES 436	
			16. PRICE CODE A19	
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT	